Stage Liesse Python

22 mai 2013

Jean-Marc ROUSSEL

jean-marc.roussel@lurpa.ens-cachan.fr

DGM / LURPA, ENS Cachan

Attentes et objectifs du stage

Votre attente

- Savoir comment enseigner l'**informatique** en CPGE
  - Algorithmique et programmation : langage retenu **Python**

Mon objectif

- Vous faire découvrir **le langage Python et ses nombreuses possibilités**
  - Python : langage très bien adapté aux besoins d'un scientifique
    - Langage objet simple disposant de nombreux modules spécialisés
- Vous faire profiter de ma propre expérience
  - **Recherche** : Chercheur au LURPA au sein de l'Équipe ISA
    - Tous nos développements sont en Python depuis de nombreuses années
  - **Enseignement** : Module **Informatique** de la formation **Saphire** à l'ENS Cachan **Sciences Appliquées en PHysique et Ingénierie pour la Recherche et l'Enseignement**
    - 6h Cours, 6h TD et un projet de 12h : langage retenu **Python**

Organisation de la journée

Hypothèses de travail

- Vous disposez peut-être que d'une expérience minime en programmation
  - **Formation sans prérequis technique**
- Vous allez devoir enseigner l'**informatique** en CPGE à la rentrée 2013
  - Il vous faut être opérationnel très rapidement...
    - Maitrise des bases du langage Python
    - Connaissances d'un environnement de programmation

Organisation retenue

- Alternance de **Présentations** et de **Manipulations pratiques**
  - La maîtrise d'un langage de programmation ne s'obtient que par la pratique

Mon objectif

- Vous permettre de développer rapidement vos propres codes Python

Programme de la formation

Premier pas avec Python

- Les caractéristiques du langage Python
- Les types de base : nombre, chaine de caractères, liste, ensemble, dictionnaire, tuple
- Les instructions composées : possibilités offertes, syntaxe
- Les fonctions : possibilités offertes, syntaxe

Introduction à la Programmation Objet

- Limites de la programmation impérative
- Principes de la programmation Objet
- Apport du diagramme de Classes d'UML en phase d'Analyse et Conception
- Du diagramme de Classes au code Python

Présentation de quelques modules additionnels

- Pour le calcul numérique : NumPy, SciPy, Matplotlib
- Pour créer des interfaces graphiques : Tkinter

Pourquoi une formation en Informatique en Saphire ?

Omniprésence des outils informatiques en sciences appliquées

- Outils propriétaires de plus en plus ouverts :
  - Import/Export de données
  - Développement de modules spécifiques
- Outils « Open Source » développés par la communauté scientifique
  - Aucune limite de personnalisation...

Besoin grandissant de développements logiciels personnalisés

- Codage d'algorithmes spécifiques pour des travaux de recherche
- Interfaçage d'outils à travers l'export puis l'import de données
- Automatisation d'expérimentations et de la mise en forme de données

Tout jeune scientifique doit avoir un minimum de culture informatique lui permettant de développer des maquettes logicielles.

Qu'apporte ce module aux étudiants de Saphire ?

Du travail

- Feuille de TD : au moins 12 heures...
- Projet : au moins 18 heures...

Des déboires...

- Au moins une centaine de messages d'erreur...
  - des erreurs de syntaxe
  - des erreurs à l'exécution
- Des résultats incohérents...

De la satisfaction...

- lorsque **leurs propres codes** tournent.

Une initiation au langage Python

- Un langage reconnu pour sa puissance et sa simplicité d'utilisation

Une aide méthodologique

- Introduction à la programmation objet
  - De la spécification en UML à l'implantation en Python

Sur le plan culturel

- Introduction à l'analyse syntaxique
  - Comment retrouver les données pertinentes contenues dans un texte ?

une aide pratique

- Des conseils pour vous éviter de refaire les traditionnelles erreurs...

Détails des 3 problèmes traités en TD (Formation Saphire)

Problème 1 : Tour de Hanoï

- Établir un programme qui liste les 127 mouvements nécessaires au déplacement d'une tour aux 7 couleurs de l'arc-en-ciel.

Problème 2 : Le jeu du pendu

- Établir un programme qui vous fasse découvrir un mot pris au hasard dans un fichier texte formaté (un mot par ligne)

Problème 3 : Fraction

- Développer votre propre module pour le calcul avec des fractions
  - Addition, Soustraction, ...
  - Tests comparatifs

Notions abordées

- Primitives de base de Python
- Instruction composées, Fonctions
- Affichage de messages
- Manipulation de chaines de caractères
- Gestion de fichiers
- Interaction clavier
- Programmation objet

Programmation informatique

Ensemble des activités qui permettent l'obtention d'un code informatique

Activité essentielle du **Génie Logiciel**

trop rarement enseignée

Analyse

- **Spécification** : Définition du but programme
  - Que doit faire le programme ?
  - Que produit-il ? A partir de quelles données ?
- **Conception** : Définition du comment
  - Étude simultanée des données manipulées et de l'algorithme de traitement

Codage

- Implantation de l'algorithme à l'aide d'un langage de programmation

Test du programme

- Détection des erreurs

Langage python

Pourquoi Python ?

- **Langage gratuit et de source libre**
  - Importante documentation en ligne
- **Langage simple à mettre en œuvre**
  - Primitives de base
  - Typage dynamique
  - Gestion automatique de la mémoire
  - Analyse interactive de la structure de données
- **Langage Orienté Objet**
  - Programmation simplifiée
- **Langage porté par la communauté**
  - Modules complémentaires
    - Mathématiques, graphiques, ...

Source

- http://www.python.org/
  - Version : **Python 3.3.2**
  - Plateformes **: Windows, Mac, Linux...**
    - La version Windows intègre un éditeur (**IDLE**) simple à utiliser avec une aide en ligne accessible par la touche **F1**

Documentation en français

- **Introduction à Python 3**
  - Bob CORDEAU (IUT d'Orsay)
- **Apprendre à programmer avec Python 3**
  - Gérard Swinnen (Université de Liège)
- **Abrégé Dense Python 3.2**
  - Laurent Pointal (Ing. Etudes LIMSI)

De nombreux exemples présentés dans ce support sont tirés de ces documents.

Les variables

Définitions

- **Concept** :
  - Nom donné à une valeur.
- **Informatiquement** :
  - Référence à une adresse mémoire.

Techniques d'affectation

```
maVar = 4       # forme de base
a = maVar*3     # forme de base
a += 2          # a = a + 2
c = d = 8       # cibles multiples
e, f = 2.7, 5   # affectation par position
e, f = f, e     # échange e et f
print(a,e)      # Affichage
  14 5
```

Noms des variables

- Différents des mots réservés Python.
- A base de lettres, chiffres et _

Conseils de nommage

- Commencer par une minuscule ou _
  - Règle d'usage Python
- Privilégier la relecture du code

Typage des variables

- Python détermine automatiquement le type de la variable lors de l'affectation.

Python et les types de données

Les types de base

- **Nombres** : entier, flottant, complexe

```
a = 3.14      # flottant
b = 3.1e5     # 3.1 x 10^5
c = 3+5j      # Complexe
```

- **Chaînes de caractères**

```
c1='Saphire' # Série de caract.
c2='Notes :\n- Toto\t15/20,\n'
```

- **Listes** : Collection ordonnée d'éléments
  - Doublons possibles
- **Ensembles** : Collection non ordonnée d'éléments sans doublons
- **Dictionnaires** : Collection de couples « clé : valeur » (unicité des clés)
- **Tuples** : Collection ordonnée d'éléments
  - Non modifiable une fois définies

Caractéristiques

- Chaque type dispose de fonctions prédéfinies pour les manipuler
  - Composition, parcours, tri
  - Recherche d'éléments
  - ...

Vous pouvez définir vos propres types de données : Les objets

```
f1=Fraction(1,3) # Création
f2=Fraction(2,3) # Création
f3=f1+f2
print(f1,f2,f3)  # Affichage
  1/3 2/3 1
```


Les chaînes de caractères (1)

Trois notations disponibles

- Guillemets pour inclure facilement des apostrophes

```
c1="L'eau vive"
```

- Apostrophes pour inclure facilement des guillemets

```
c2=' est "froide" !\n'
```

- Triples guillemets pour conserver la mise en page (lignes multiples) :

```
c3 = """Usage :
-h : help
-q : quit"""
```

Caractéristiques

- Ne sont pas modifiables
- Peuvent être composées entre-elles avec : +, *

```
c4=c1+c2+'-*-'*3   # 'L\'eau vive est "froide" !\n-*--*--*-'
```

- Disposent de nombreuses méthodes pour les manipuler

Les chaînes de caractères (2)

Méthodes associées

- **Mise en forme**
  - capitalize, center, expandtabs, **format**, ljust, lower, lstrip, title, swapcase, upper
- **Test de format**
  - endswith, isalnum, isalpha, isdigit, islower, isspace, istitle, isupper, startwith
- **et toutes les autres...**
  - count, decode, encode, find, index, **join**, partition, **replace**, rfind, rindex, rjust, rpartition, rsplit, rstrip, split, splitlines, strip, translate, zfill

Exemple

```
c1="L'eau vive"
c2=c1.upper()          # "L'EAU VIVE"
c3=c1.replace('e','E') # "L'Eau vivE"
```


Les listes (1)

Définition

- Collection hétérogène, ordonnée et modifiable

```
maListe = ['a', 2, [1, 3], 'bc', 2, 5]
couleurs = ['trèfle', 'carreau', 'coeur', 'pique']
couleurs[0]  # premier élément de la liste
couleurs[-2] # deuxième élément en partant de la fin
maListe[2:4] # sous-liste de maListe
      [[1, 3],'bc',2]
```

Caractéristiques

- Sont modifiables `maListe.append('Saphire')`
- Disposent de nombreuses méthodes pour les manipuler
  - **append**, count, extend, index, insert, pop, remove, reverse, sort

Les listes (2)

Attention : les listes sont manipulées par leur adresse

```
joursOuvres=['lundi','mardi','mercredi','jeudi','vendredi']
joursSemaine=joursOuvres
joursSemaine.extend(['samedi','dimanche'])
print (joursSemaine)
  ['lundi','mardi','mercredi','jeudi','vendredi',
  'samedi','dimanche']
print (joursOuvres)
  ['lundi','mardi','mercredi','jeudi','vendredi',
  'samedi','dimanche']
```

Correction

```
joursOuvres=['lundi','mardi','mercredi','jeudi','vendredi']
joursSemaine=joursOuvres[:]
joursSemaine.extend(['samedi','dimanche'])
```

Création d'une copie

Les ensembles

Définition

- Collection hétérogène **non ordonnée** d'éléments sans doublons

```
ens1=set(['lundi', 'lundi', 'mercredi', 'jeudi'])
ens2={'mercredi','jeudi','vendredi'}
```

Caractéristiques

- Sont modifiables.
- Disposent de nombreuses méthodes pour les manipuler :
  - add, clear, copy, difference, difference_update, discard, intersection, intersection_update, issubset, issuperset, pop, remove, symmetric_difference, symmetric_difference_update, union, update

```
ens3=ens1.union(ens2)
print(ens3)
  {'jeudi', 'lundi', 'vendredi', 'mercredi'}
```

Les dictionnaires

Définition

- Collection de couples « clé : valeur »

```
monDico={'je':'I', 'vous':'you'}  # Définition
monDico['tu']='du'    # Ajout d'un nouveau couple
monDico['tu']='you'   # Modif. de la valeur associée à la clé 'tu'
del(monDico['tu'])    # suppression d'un couple
```

Caractéristiques

- Structure **non ordonnée** dont les clés sont uniques
- Disposent de nombreuses méthodes pour les manipuler
  - clear, copy, fromkeys, get, items, keys, pop, popitem, setdefault, update, values

Les Tuples

Définition

- Collection hétérogène, ordonnée et **non modifiable** d'éléments

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

Caractéristiques

- Ne sont pas modifiables
- S'utilisent comme les listes mais leur parcours est plus rapide

Rôle

- Structure de données connues à l'avance :
  - Coordonnées d'un point, …

Les instructions composées

Actions

- Choisir : **if - [elif] - [else]**
- Boucler : **while**
- Parcourir : **for ... in ...**

Tabulation

```
if x>0 and x<10 :
    print('Positif inférieur a 10')
elif x<20 :
    print('Positif inférieur a 20')
else :
    print('Plus grand que 20')
```

Syntaxe

- Une ligne d'en-tête terminée par **:**
- Un bloc d'instructions indenté par rapport à la ligne d'en-tête

```
x = 257
cpt = 0
print('Log en base 2 de', x, ': ',end='')
while x > 1:
    x = x//2 # division entière
    cpt += 1 # incrémentation
print(cpt,'(Valeur approchée)')
```

```
Log en base 2 de 257 : 8 (Valeur approchée)
```

Caractéristiques

- Peuvent être imbriquées, interrompues

```
for lettre in 'ciao':
    print(lettre)
for elt in [2, 'a', 3.14]:
    print(elt)
for i in range(5):
    print(i)
```


Les fonctions

Groupe d'instructions s'exécutant à la demande

- **Élément de structuration d'un programme**
  - Accepte des paramètres en entrée
    - Obligatoire ou optionnel (valeur définie par défaut)
  - Retourne une ou plusieurs valeurs

Syntaxe

Tabulation

```
def nom_fonction(parametres) :
    """Documentation de la fonction. """
    <bloc_instructions>
    return resultat
```

Caractéristiques

- Peuvent être imbriquées les unes dans les autres
- Variables définies localement

FAQ sur l'utilisation de Python en enseignement

Quelle version de Python faut-il utiliser ? (Python 2 ou Python 3)

- **Python 3** sans hésiter...
  - Il reste encore quelques modules qui n'ont pas été portés en Python 3 par la communauté.
    - Cependant, ce ne sont pas ceux dont vos étudiants ont besoin

Quelle plateforme choisir ?

- **Celle qui sera dans vos salles de TP** :
  - ENS Cachan, Formation Saphire
    - 2012/2013 : **80 étudiants** répartis en 5 groupes de TD **en parallèle**
      → 80 ordinateurs utilisés simultanément (dont bcp de machines personnelles)

Quel environnement de développement ?

- Le meilleur environnement est celui qu'on arrive à faire maitriser à nos étudiants...
  - L'environnement **IDLE** a le mérite d'être :
    - simple à utiliser,
    - automatiquement installé sous Windows,
    - offrir une aide en ligne sur Python accessible via la touche **F1**.

Utilisation de l'environnement IDLE (1)

Démarrage de IDLE

- Menu démarrer > Tous les programmes > Python 3.3 > IDLE (Python GUI).
  - La fenêtre ouverte est une console (ou shell) Python

Utilisation du shell

- Python étant un langage interprété, il est techniquement possible de composer un programme à la volée directement dans un shell, instruction par instruction.
- Le shell est classiquement utilisé pour afficher les résultats des programmes, saisir des valeurs en mode interactif...
- A l'issue de l'exécution d'un programme, il est également possible d'accéder aux données manipulées.
  - Cette fonctionnalité permet de facilement tester certaines parties du code...
    → à utiliser sans aucune restriction

Utilisation de l'environnement IDLE (2)

Exécution d'un script Python

- Il faut tout d'abord charger dans l'éditeur votre script s'il n'y est pas déjà présent par le menu File -> Open.
- Pour l'exécuter, il suffit d'appuyer sur la touche F5 (ou bien aller dans le menu Run -> Run Module).
  - L'exécution se fera dans un shell

Écriture d'un script Python

- En plus d'être un Shell interactif, IDLE est aussi un éditeur spécialement conçu pour des programmes Python. Pour ouvrir IDLE en mode éditeur, vous pouvez, à partir du mode interactif, aller dans le menu File - New Window.
- Vous pouvez alors écrire un programme Python et l'enregistrer dans un fichier d'extension .py.
- Vous pourrez ensuite ouvrir ce fichier avec IDLE en cliquant dessus avec le bouton droit de la souris -> Edit With IDLE.

Stage Liesse Python

A vos machines...

Initiation à la Programmation Objet

Ce que savez déjà...

- Primitives du langage Python

Ce que vous ne savez peut-être pas encore...

- Principe de la modélisation Objet
- Diagramme de Classes d'UML
- Implémentation d'un diagramme de Classes en Python

Programmation informatique

Ensemble des activités qui permettent l'obtention d'un code informatique

Activité essentielle du **Génie Logiciel**

trop rarement enseignée

Analyse

- **Spécification** : Définition du but programme
  - Que doit faire le programme ?
  - Que produit-il ? A partir de quelles données ?
- **Conception** : Définition du comment
  - Étude simultanée des données manipulées et de l'algorithme de traitement

Codage

- Implantation de l'algorithme à l'aide d'un langage de programmation

Test du programme

- Détection des erreurs

Les paradigmes de programmation

Définition

- **Style fondamental de programmation informatique** qui traite de la manière dont les solutions aux problèmes doivent être formulées dans un langage de programmation.

Exemples

- **Programmation impérative**
  - Approche centralisée du problème basée sur les traitements
    - Paradigme originel et le plus courant
- **Programmation objet**
  - Approche décentralisée basée sur la collaboration d'éléments autonomes
- ...

Caractéristiques

- Largement dépendant du langage de programmation utilisé
- Conditionne la manière d'appréhender un problème

Programmation impérative (1)

Principe

- Décrire les opérations à réaliser en termes de séquences d'instructions
  - l'assignation
  - l'appel à des fonctions
  - le branchement conditionnel (Si ... Alors ... Sinon ...)
  - le bouclage (Pour tout... , Tant que ...)

Technique d'analyse

- Décomposition du problème en sous-problèmes plus simple à résoudre

Élément de structuration au niveau du code

- Fonctions
  - Rôle : fournir un résultat à partir de paramètres d'entrée

Programmation impérative (2)

Avantages

- Paradigme originel, simple à comprendre
  - Basé sur une décomposition fonctionnelle et hiérarchique
    - Fonction principale, Sous-fonctions, ...
- Supporté par la majorité des langages de programmation

Inconvénients

- N'est simple à mettre en œuvre que sur des problèmes parfaitement identifiés
  - Difficultés à décomposer le problème initial
  - Résultat peu structurant : arbre d'appel de fonctions
- Très mal adaptée au traitement de données fortement interconnectées
- Ne permet pas de résoudre les problèmes à l'aide de techniques récursives

Programmation objet (1)

Origine

- **Échec des méthodes traditionnelles** lors du déploiement de gros logiciels
  - Phase d'analyse et/ou conception trop longue due à la difficulté d'appréhender le problème dans sa globalité
  - Structure du logiciel basée principalement sur les fonctions à réaliser
    - Une évolution fonctionnelle peut entrainer des modifications structurelles lourdes
  - Tests et réutilisabilité du logiciel difficile à mettre en œuvre

Caractéristiques de l'Approche Objet

- **Système à réaliser** : **Un tout organisé**, dont les éléments solidaires peuvent être définis les uns par rapports aux autres.
- **Méthode d'analyse** : Intégration de deux points de vue complémentaires
  - la constitution : Quels sont les éléments qui composent le système ?
  - les différents rôles : Que fait le système ? Que fait chaque composant ?

Programmation objet (2)

Principe de base

- Concevoir le programme en tant qu'assemblage de briques logicielles qui interagissent entre-elles : les **OBJETS**
  - **Objet** : Composant **autonome** disposant :
    - de données qui lui sont propres,
    - de comportements élémentaires.

Intérêts de la méthode

- Stabilité de la modélisation par rapport aux entités du monde réel
  - Objet informatique : **abstraction** d'une entité du monde réel
- Construction itérative facilitée par le faible couplage entre objets
- Possibilité de réutilisation des objets

Programmation objet (3)

A quoi correspondent les objets ?

- Une entité du monde physique
- Un concept, une idée

Précepte de modélisation :
Les objets pertinents sont ceux que l'utilisateur identifie naturellement.

Que fait un objet ?

- Il est créé, il évolue et peut disparaître.
- Il dispose d'**attributs** qui le caractérisent.
- Il dispose de **méthodes** lui permettant d'interagir sur ses attributs et sur son environnement.

Comment est-il défini ?

- C'est l'**instance** d'une classe.
  - **Classe** : description abstraite d'un ensemble d'objets présentant les mêmes caractéristiques.

Modélisation d'un jeu d'échecs

Principe de modélisation

- **Identifier les objets du discours**
  - Éléments réels : Pièces, Joueurs, ...
  - Concepts : Parties, Coups, ...
- **Rechercher leurs caractéristiques**
  - En terme d'attributs
  - En terme de méthodes

- **Identifier les interactions entre les objets**
  - Une partie oppose deux joueurs.
  - Deux pièces ne peuvent pas avoir la même position.

Technique de travail

- Matérialiser votre analyse à l'aide d'un **diagramme de classes** (UML)

Extrait de :
UML par la pratique
Pascal Roques
Eyrolles

UML : Unified Modeling Language

Qu'est-ce ?

- Langage de modélisation graphique à base de pictogrammes
  - 13 types de diagrammes
- Modélisation d'un projet informatique tout au long de son cycle de vie.

Diagrammes structurels

- **Diagramme de classes**
- Diagramme d'objets
- Diagramme de composants
- Diagramme de déploiement
- Diagramme des paquetages
- Diagramme de structure composite

Diagrammes comportementaux

- Diagramme des cas d'utilisation
- Diagramme états-transitions
- Diagramme d'activité

Diagrammes dynamiques

- Diagramme de séquence
- Diagramme de communication
- Diagramme global
- Diagramme de temps

UML : Unified Modeling Language

Intérêts des diagrammes

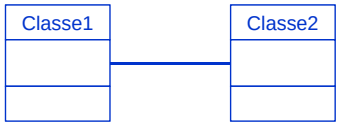
- Formalisation des besoins et de la solution mise en œuvre
  - Exprimer une idée oblige d'y réfléchir avec précision...
- Faciliter l'analyse et la conception d'un logiciel en travaillant sur des représentations graphiques abstraites du logiciel à concevoir
  - Définition du problème à un haut niveau sans rentrer dans les spécificités
- Permettre les discussions autour des diagrammes construits
  - Un schéma vaut mieux qu'un long discours...
- Génération automatique de squelette de code

Le diagramme de classes d'UML

Constitution

- **Classes** : description abstraite d'un ensemble d'objets
  - Nom, Attributs, Méthodes

- **Relations** entre les classes
  - Association
    - Étiquette : Nature
    - Cardinalités : Nbre d'objets interagissant
    - Rôles : Informations complémentaires
    - Restriction de la navigabilité



  - Agrégation / Composition

  - Héritage

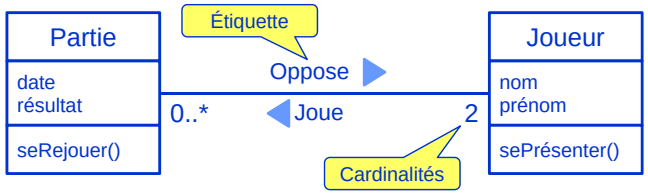
Relation entre classes

Qu'est-ce ?

- association sémantique durable entre au moins deux classes

Leurs rôles

- Matérialiser les interactions entre les différentes classes
  - Une partie oppose deux joueurs.



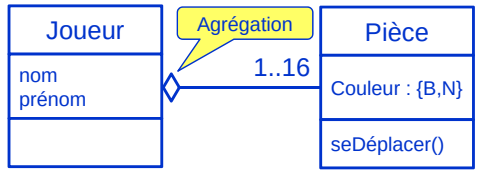
Multiplicités (Cardinalités)

1 : la classe est en relation avec un et un seul objet de l'autre classe.
1..* : la classe est en relation avec au moins un objet de l'autre classe.
0..* : la classe est en relation avec 0 ou n objets de l'autre classe.
0..1 : la classe est en relation avec au plus un objet de l'autre classe.

Agrégation / Composition

Qu'est-ce ?

- Association entre deux classes dont l'une joue un rôle prépondérant vis-à-vis de l'autre.
  - Un joueur dispose de pièces.
  - Une partie se décompose en coups.

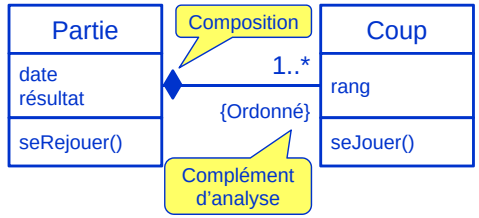


Agrégation

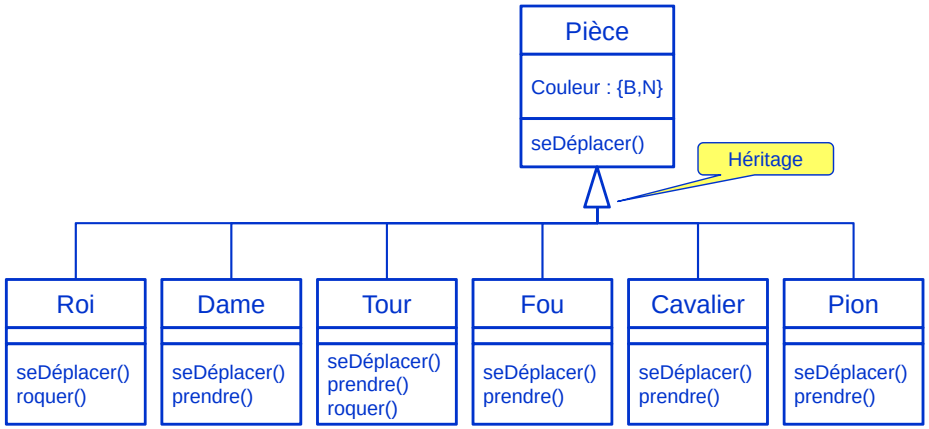
- Cas général

Composition

- Cas particulier de l'agrégation
  - Unicité de l'agrégat
  - Destruction du composant si destruction de l'agrégat



Relation d'héritage



Caractéristiques de l'héritage

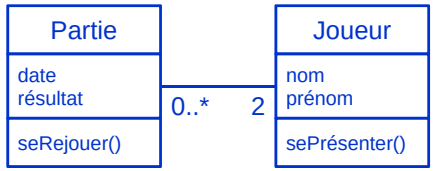
- Généralisation de l'analyse en créant des superclasses (abstraites ou non)
- Héritages multiples possibles
- Polymorphisme : Spécialisation dynamique de certaines méthodes

Comment identifier les relations ?

Conseils

- Formaliser toutes les informations données dans le cahier des charges
- Définir précisément chaque relation
  - Cardinalités
  - Rôles
  - Contraintes induites par les relations

Proposition 1



Proposition 2

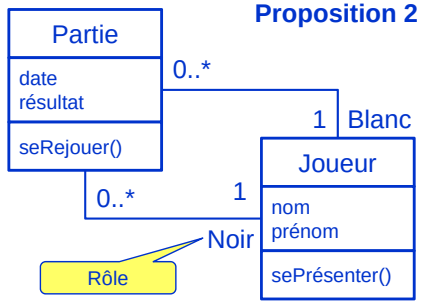
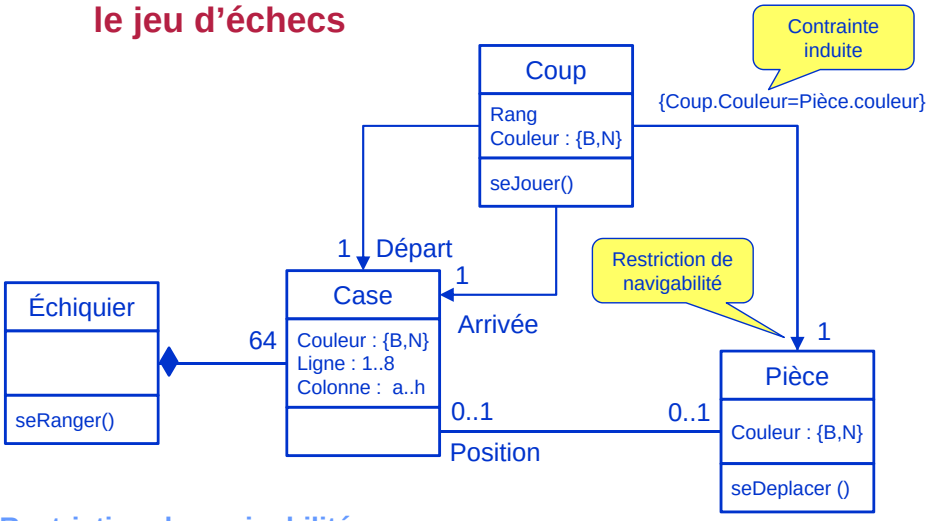


Illustration pour le jeu d'échecs



Restriction de navigabilité

- Précise la non symétrie d'une association

Implantation d'un diagramme de Classes UML à l'aide du langage Python

Objectif

- Matérialisation du travail de conception en une structure de code Python

Diagramme de Classes UML → Programme Python

Principe

- Une Classe UML → Une Classe Python
  - Un Attribut → Variable associée : nom, valeur
  - Une Méthode → Fonction associée
- Un héritage UML → Un héritage Python
- Une relation UML → Des compléments dans les classes Python (Variables et fonctions)
  - Cardinalités
  - Navigabilité
  - Rôles

Classes et objets en Python

Concept

- **Classe** : description globale d'un ensemble d'objets présentant les mêmes caractéristiques.
- **Objet** : **instance** d'une classe.

Programme

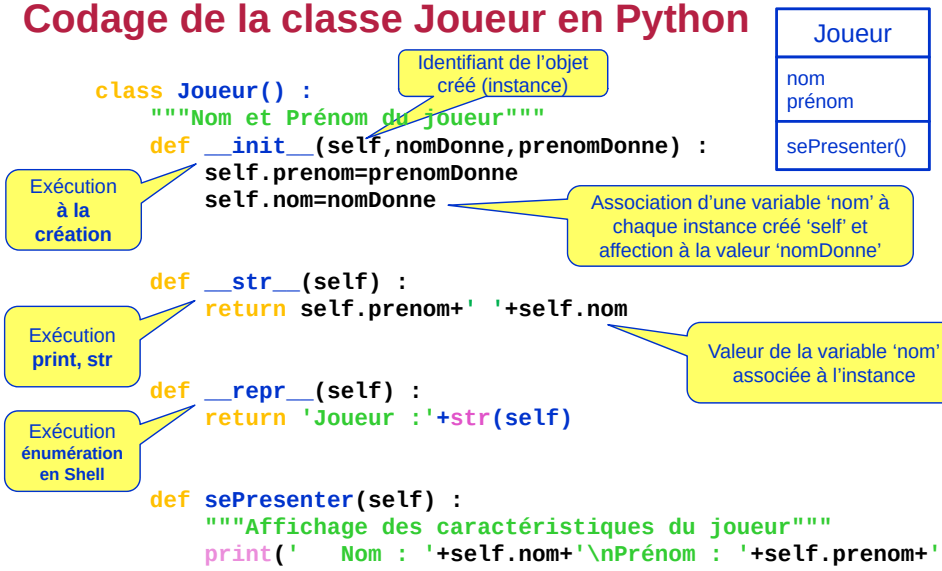
- Utilisation d'objets ayant des données propres, sachant interagir entre-eux.

```
f1=Fraction(1,3) # Création d'une instance
f2=Fraction(2,3) # Création d'une instance
f3=f1+f2
print(f1,f2,f3)  # Affichage
      1/3 2/3 1
```

Définition d'une classe

- Définition des différentes méthodes qu'utiliseront les objets pour interagir
  - Méthodes **génériques** : création, affichage, énumération, destruction
  - Méthodes **spécifiques** à la classe :
    - Fraction : addition, multiplication, comparaison

Codage de la classe Joueur en Python

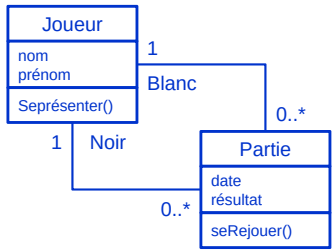



```
class Joueur() :
    """Nom et Prénom du joueur"""
    def __init__(self,nomDonne,prenomDonne) :
        self.prenom=prenomDonne
        self.nom=nomDonne

    def __str__(self) :
        return self.prenom+' '+self.nom

    def __repr__(self) :
        return 'Joueur :'+str(self)

    def sePresenter(self) :
        """Affichage des caractéristiques du joueur"""
        print('   Nom : '+self.nom+'\nPrénom : '+self.prenom+'\n')
```

Codage d'une relation

Relation → Des variables supplémentaires dans les classes

- **Influence de la navigabilité**
  - bidirectionnelle:
    - une variable dans chaque classe
  - unidirectionnelle :
    - Pas de variable dans la classe extrémité
- **Influence des cardinalités**
  0..1 : Simple variable
  0..***** : Variable de type Liste ou ensemble




```
class Joueur() :
    def __init__(self,nom,prenom) :
        ...
        self.partiesJouees=[]

    def ajouterPartie(self,partie) :
        self.partiesJouees.append(partie)
```

```
class Partie() :
    def __init__(self,joueur1,joueur2) :
        ...
        self.joueurBlanc=joueur1
        self.joueurNoir=joueur2
        joueur1.ajouterPartie(self)
        joueur2.ajouterPartie(self)
```


FAQ au sujet du diagramme de Classes

Nécessité du diagramme ?

- L'analyse doit être faite, autant le faire le plus proprement possible...

Unicité du modèle ?

- Presque jamais ☹

Description des méthodes

- Ne fait pas l'objet du diagramme des classes

Aspect Public/Privé des classes ?

- Mécanisme non implanté dans Python

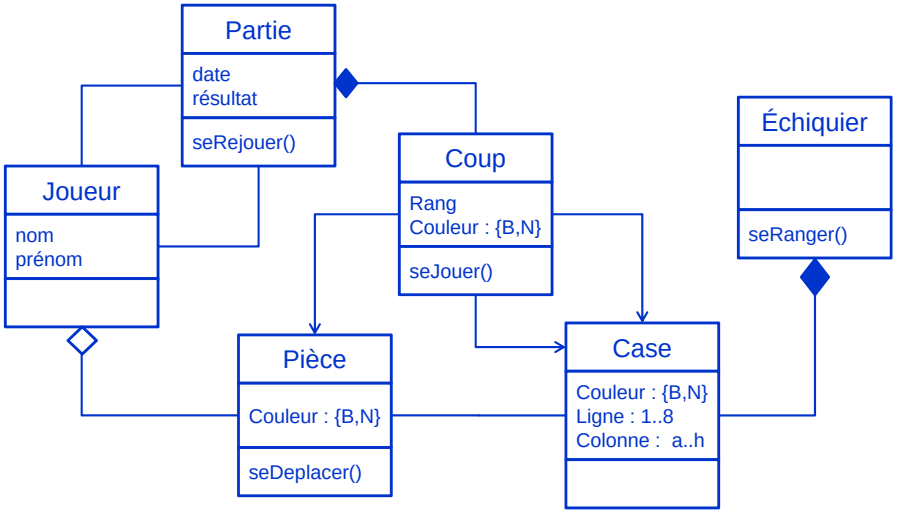
Complétude du modèle ?

- Classes :
  - Analyse du besoin et des entités manipulées
- Relations :
  - Formalisation des données du CdC
  - Par simulation des méthodes en testant les échanges avec les autres objets

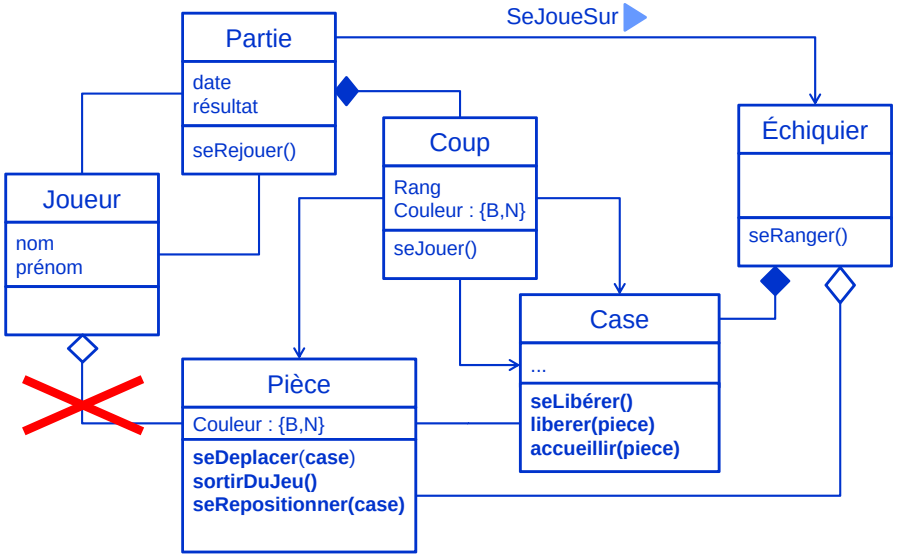
A quoi ressemble un code Objet ?

- Une liste de classes
  - Une liste de méthodes
- Une séquence d'opérations sur des objets
  - Création d'objet, animation d'objet

Analyse du modèle établi par simulation



Premiers enrichissements



Détails des méthodes (1)

```
class Partie() :
    def seRejouer(self) :
        self.echiquier.seRanger()
        for coup in self.coups :
            coup.seJouer()

class Echiquier() :
    def seRanger(self) :
        for case in self.cases :
            case.seLiberer()
        for piece in pieces :
            piece.seRepositionner(self.positionsInitiales[piece])

class Coup() :
    def seJouer(self) :
        self.piece.seDeplacer(self.caseArrivee)
```


Détails des méthodes (2)

```
class Piece() :
    def sortirDuJeu(self) :
        self.position=None
    def seRepositionner(self,case) :
        case.accueillir(self)
        self.position=case
    def seDeplacer(self,futurePosition) :
        self.position.liberer(self)
        futurePosition.accueillir(self)
        self.position=futurePosition

class Case() :
    def seLiberer(self) :
        if self.piecePresente!=None :
            self.piecePresente.sortirDuJeu()
            self.piecePresente=None
    def accueillir(nouvellePiece) :
        self.seLiberer()
        self.piecePresente=nouvellePiece
```


Conclusions

Approche Objet

- **Objet** : Composant **autonome** disposant :
  - de données qui lui sont propres,
  - de comportements élémentaires.
- Les objets interagissent entre eux via des échanges de messages.

Intérêts de l'approche Objet

- Calquée sur la réalité : Objet représentation abstraite d'un objet du réel
- Cycle de développement en spirale
  - Analyse, conception, codage, test
  - Ajout de fonctionnalités
- Réutilisation de code

Difficultés d'une conception objet

- Identification claire des interactions objet
  - Ne pas hésiter à gribouiller un diagramme de classes
- Maitrise de la décentralisation des actions
  - Solution : Diagramme de séquence UML