Labo d'introduction à l'informatique

pour les mathématiques

Yann Thorimbert

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

Semaine 8

Dictionnaires

La fin de la programmation ?

Avec les structures de contrôle que nous avons vues, ainsi que le fait de pouvoir définir différents types de variables dont les tableau, nous pouvons en théorie aborder n'importe quel problème de programmation.

Nous avons même en général plusieurs façons de résoudre un problème donné.

Aujourd'hui, nous allons voir des concepts nouveaux qui nous **simplifient** la vie ou bien rendent nos programmes plus **performants**.

Un nouveau problème

Supposons que l'on veuille écrire une fonction `prix(x)` qui retourne, grâce au nom du produit `x` d'un magasin, le prix de ce produit.

Par exemple : `prix("jus d'orange")` retourne la valeur `1.5`.

À quoi devrait ressembler le code d'une telle fonction ?

On suppose qu'il n'y a que 5 produits possibles pour l'exemple : *jus d'orange, eau minérale, soda, chips, chocolat*… mais il faut s'en imaginer davantage pour saisir l'essence du problème !

Solution à base de conditions `if/elif/else`

```
def prix(x):
    if x == "jus d'orange":
        p = 1.5
    elif x == "eau minérale":
        p = 0.75
    elif x == "soda":
        p = 2.5
    elif x == "chips":
        p = 3.5
    elif x == "chocolat":
        p = 2
    else:
        p = 0
        print("Erreur : produit non disponible")
    return p
```

Fastidieux, répétitif !

Solution à base de tableaux

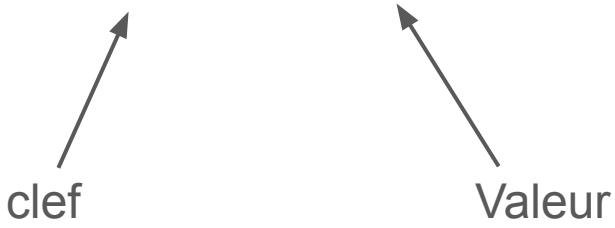
```
def prix(x):
    clefs = ["jus d'orange", "eau minérale", "soda", "chips", "chocolat"]
    valeurs = [1.5, 0.75, 2.5, 3.5, 2]
    p = 0
    for i in range(len(clefs)):
        if clefs[i] == x:
            p = valeurs[i]
            break
    if p == 0:
        print("Erreur : produit non disponible")
    return p
```

Nous verrons en cours pourquoi cette solution n'est pas performante en général.

Un nouveau type de donnée

Le plus pratique serait de pouvoir définir un "catalogue" des prix.

Un tel catalogue lierait des textes à des prix.



Une structure de donnée permettant de définir des **couples clef-valeur** est le **dictionnaire** (ou map). Le type Python correspondant est `dict`.

Déclaration et modification d'un dictionnaire

Les accolades et l'usage des deux points permettent la déclaration :

```
mon_dico = {"pomme":2, "banane":3, "poire":2.5}
```

Les crochets permettent d'obtenir la valeur associée à une clef :

```
mon_dico["pomme"] = 3 #on change la valeur associée à "pomme"
mon_dico["kiwi"] = 5 #on ajoute un nouveau couple
```

Utilisation du mot-clef *in*

Les expressions utilisant le mot-clef in permettent d'effectuer un test d'appartenance :

```
mon_dico = {"pomme":2, "banane":3, "poire":2.5}

if "banane" in mon_dico:
    print("Il y a des bananes dans le dictionnaire")

x = "pain" in mon_dico
print(x) #affiche False
```

Solution à base d'un dictionnaire

```
def prix(x):
    catalogue = {"jus d'orange": 1.5,
                 "eau minérale": 0.75,
                 "soda": 2.5,
                 "chips": 3.5,
                 "chocolat": 2}
    if x in catalogue:
        return catalogue[x]
    else:
        print("Erreur : produit non disponible")
        return 0
```

On lie des chaînes de caractères à des nombres

Solution à base d’un dictionnaire (alternative)

```
catalogue = {"jus d'orange": 1.5,
             "eau minérale": 0.75,
             "soda": 2.5,
             "chips": 3.5,
             "chocolat": 2}

#Avec la méthode get, on peut spécifier une valeur par défaut
prix = catalogue.get(x, 0)
```

Les avantages de la solution avec le dictionnaire

- **Clarté** du code.
- **Performance** du code pour l'accès aux éléments.
- Une variable unique pour tout **regrouper**.
- Permet de facilement **insérer** ou **retirer** des valeurs, y compris durant l'exécution :
  - Insertion d'un couple clef-valeur supplémentaire :
    `catalogue["riz"] = 3.5`
  - Suppression d'un couple clef-valeur existant :
    `catalogue.pop("soda")`

Quelques nuances

- Une clef possède une seule valeur, mais une valeur donnée peut correspondre à de multiples clefs.
- Les valeurs peuvent être n'importe quel objet Python.
- Les clefs peuvent être des `str`, des `int`, des `float`, …
- … mais les clefs doivent pouvoir être "hachables" (cf. un prochain cours théorique).
- En Python, tous les objets built-in (présents de base dans le langage) immutables sont également hachables.

Itérer sur les clefs d’un dictionnaire

Par défaut, Python itère sur les clefs d'un dictionnaire :

```
mon_dico = {"pomme":2, "banane":3, "poire":2.5}

for fruit in mon_dico:
    print(f"Le fruit {fruit} coûte {mon_dico[fruit]} chf")
```

Notez que ceci équivaut à :

```
for fruit in mon_dico.keys():
    ...
```

Itérer sur les couples d’un dictionnaire

On peut itérer sur les couples clefs-valeurs :

```
mon_dico = {"pomme":2, "banane":3, "poire":2.5}

for fruit, prix in mon_dico.items():
    print(f"Le fruit {fruit} coûte {prix} chf")
```

Itérer sur les valeurs d'un dictionnaire

Parfois, on peut être intéressé par les valeurs uniquement :

```
mon_dico = {"pomme":2, "banane":3, "poire":2.5}

tot = 0
for prix in mon_dico.values():
    tot += prix
print("Le total vaut", tot)
```

Exercice rapide

Considérons un dictionnaire d qui permet de trier des mots dans trois catégories. Si le mot ne contient pas la lettre "e", on dira que c'est un mot "nul". Si moins de 30% de ses lettres sont des "e", on dira que c'est un mot "normal". Sinon, on dira que c'est un mot "excessif".

La fonction suivante permet d'ajouter automatiquement un mot au dictionnaire.

```
def ajout(d, mot):
    nb_de_e = mot.count("e") # On utilise la méthode count du type str
    ... # À vous de trouver le code ici
```

Exercice rapide - Correction

```
def ajout(d, mot):
    nb_de_e = mot.count("e") # On utilise la méthode count du type str
    if nb_de_e == 0:
        d[mot] = "nul"
    elif nb_de_e/len(mot) < 0.3:
        d[mot] = "normal"
    else:
        d[mot] = "excessif"

mes_mots = ["banane", "exemple", "eleve", "test", "epee", "paix", "rhythme"]
mon_dico = {}
for m in mes_mots:
    ajout(mon_dico, m)
print(mon_dico)
```

Quelle solution choisir ?

- Dépend du problème.
- Toutes les solutions sont fonctionnelles, mais toutes ne sont pas efficaces ou faciles à implémenter.
- Savoir choisir la solution pertinente fait partie du travail de programmation.
- Dans certaines situations, plusieurs solutions se valent.

Exemples de solutions naturelles

- Stocker les coefficients d'un polynôme : liste de nombres.
  `coeffs = [3, 0, -6, 5]`

- Stocker différents mots : liste de chaînes de caractères.
  `mots = ["Ballon", "Balle", "Frisbee"]`

- Stocker l'altitude de différentes villes : dictionnaire.
  `altitudes = {"Genève":400, "Barcelone":0, "La Paz":3650}`

Encore un nouveau problème

- Stocker les coefficients d'un polynôme : tableau de nombres.
- Stocker différents mots : tableau de chaînes de caractères.
- Stocker l'altitude de différentes villes : dictionnaire.
- **Stocker la population, pays, indicatif téléphonique de différents villes ?**

Encore un nouveau problème

- Stocker l'altitude, population, pays, indicatif téléphonique de différents villes ?
  ⇒ On peut imaginer toutes sortes de solutions.

- Exemple de solution possible :

```
nom = ['Genève', 'Barcelone', 'New York', 'Tokyo']
pop = [5e5, 2e6, 8e6, 14e6]
pays = ['Suisse', 'Espagne', 'USA', 'Japon']
etc…
```

  ⇒ mais dans tous les cas, cela constitue une solution *ad hoc*.
  Il est désirable de tout **regrouper** au sein d'une **variable unique** !

- Dans cette situation, déclarer une nouvelle **structure de données** peut être utile.

Structures/Classes

- Structure : permet d’associer un nombre arbitraire de données à une variable unique.
- Certains langages (comme Python) permettent de définir des classes, qui sont des structures enrichies de fonctionnalités/comportements supplémentaires.
- Les différents champs de données associés à une classe se nomment les **attributs**.
- Les différentes fonctions associées à une classe se nomme les **méthodes**.
- Une classe est un peu un "patron" ou "chablon" permettant de définir un **type** de donnée. Lorsqu'un objet d'une certaine classe est effectivement créé, on dit que c'est une "**instance**" de ladite classe.

Structures/Classes

- L'étude des langages **orientés-objets** est une matière à part entière de l'informatique (pas le but de ce cours).
- En Python, on retrouve les objets un peu partout, mais le langage ne colle pas entièrement à la philosophie de l'orienté-objet pur et dur (e.g. comme Java).
- On peut très bien utiliser les objets d'une classe sans jamais définir de nouvelle classe soi-même. C'est ce que nous faisons depuis le début avec tous les objets Python (flagrant surtout avec `str`, `list` et `dict`). Référez-vous aux indications du test Moodle pour plus de précisions.