Première NSI
Chapitre X - Les dictionnaires

I. Généralités

I.1. Définition

Définition

Un dictionnaire est une collection non ordonnée mutable d'éléments. Chacun des éléments est une paire (*clé* : *valeur*). Chaque clé doit être unique.

Rémarque

En Python, on écrit les éléments entre accolade séparés par une virgule.

Propriété

Pour ajouter un élément à un dictionnaire, il faut indiquer la clé et la valeur associée.

Exemple

```
>>> mon_dico = {1: 'Mercure', 'deuxième': 'Vénus', 3: 'Terre', 'étoile': 'Soleil'}
>>> mon_dico[1]
'Mercure'
>>> mon_dico['étoile']
'Soleil'
>>> mon_dico[2]
Traceback (most recent call last):
  ...
KeyError: 2
>>> mon_dico[4] = 'Mars'
>>> mon_dico
{1: 'Mercure', 'deuxième': 'Vénus', 3: 'Terre', 'étoile': 'Soleil', 4: 'Mars'}
```

I.2. Méthodes

I.2.a. La méthode keys()

Définition

La méthode `.keys()` renvoie un itérable dont les éléments sont les clés du dictionnaire.

Exemple

```
>>> mon_dico = {1: 'Mercure', 'deuxième': 'Vénus', 3: 'Terre', 'étoile': 'Soleil'}
>>> for element in mon_dico.keys():
        print(element)
1
'deuxieme'
3
'étoile'
```

I.2.b. La méthode values()

Définition

La méthode `.values()` renvoie un itérable dont les éléments sont les valeurs du dictionnaire.

Exemple

```
>>> mon_dico = {1: 'Mercure', 'deuxième': 'Vénus', 3: 'Terre', 'étoile': 'Soleil'}
>>> for element in mon_dico.values():
        print(element)
'Mercure'
'Vénus'
'Terre'
'Soleil'
```

I.2.c. La méthode items()

Définition

La méthode `.items()` renvoie un itérable dont les éléments sont les couples $(clé, valeur)$.

Exemple

```
>>> mon_dico = {1: 'Mercure', 'deuxième': 'Vénus', 3: 'Terre', 'étoile': 'Soleil'}
>>> for element in mon_dico.items():
        print(element)
(1, 'Mercure')
('deuxième', 'Vénus')
(3, 'Terre')
('étoile', 'Soleil')
```

I.3. Les fonctions

I.4. La fonction dict()

Définition

La fonction `dict()`, si elle n'a pas de paramètres, renvoie un dictionnaire vide.
Si elle a des paramètres, ces derniers doivent être du type $cl = valeur$, où cl commence par un caractère alphabétique. Auquel cas, elle renvoie un dictionnaire ayant pour clés et valeurs les paramètres donnés.

Exemple

```
>>> mon_dico = dict{}
>>> mon_dico
{}
>>> mon_dico = dict(nom = 'caneri', profession = 'enseignant')
>>> mon_dico
{'nom': 'caneri', 'profession': 'enseignant'}
```

I.5. La fonction del()

Définition

La fonction `del()` supprime une référence (un couple $(cl : valeur)$) du dictionnaire.

Exemple

```
>>> mon_dico = dict(nom = 'caneri', profession = 'enseignant')
>>> del(mon_dico['nom'])
>>> mon_dico
 {'profession': 'enseignant'}
```

II. Les p-uplets nommés

Définition

Les p-uplets nommés est un p-uplet dont un élément est appelé par un descripteur plutôt que par un indice.

Remarque

En Python, les p-uplets nommés n'existent pas nativement, donc on peut les implémenter par un dictionnaire.

Exemple

Considérons le triplet (3-uplet) `personnage = ('Summers', 'Scott', 1.85)`.

Lorsqu'on veut récupérer le prénom, il faut écrire `personnage[0]`, ce qui n'est pas explicite.
On peut alors utiliser les p-uplets nommés. Ainsi, notre triplet est du type $(nom, prnom, taille)$.
On peut implémenter ce triplet par un dictionnaire :

```
>>> personnage = {"nom": 'Summers', "prénom": "Scott", "taille": 1.85}
>>> personnage['nom']
 'Summers'
```

Récupérer le nom est donc plus explicite, puisqu'il suffit d'écrire `personnage['nom']`.

Remarque

Les p-uplets nommés sont normalement immuables mais les dictionnaires le sont. Nous nous efforceront donc de ne pas modifier ces dictionnaires.

III. Exercices

III.1. Exercice 1

1. Écrire une fonction `dec_to_hex(nb: int) -> str` qui renvoie la représentation hexadécimale du nombre entier `nb` mis en paramètre. *On utilisera les dictionnaires. On n'utilisera pas la fonction* `hex()`.

2. Écrire une fonction `hex_to_dec(chaine: str) -> int` qui renvoie la représentation dans le système décimale du nombre dont la représentation hexadécimale est `cahine`. *On utilisera les dictionnaires. On n'utilisera pas la fonction* `int()`.

III.2. Exercice 2

Dans un jeu de rôles, il y a souvent des combats.
Considérons les deux personnages suivants :

Nom	race	Force	Endurance	Points de vie
Beowulf	Humain	17	16	36
Zardoc	Orc	15	15	53

Lors d'un combat, chaque personnage attaque l'un après l'autre, le premier à attaquer étant tiré aléatoirement.
A chaque attaque, l'attaquant lance un dé à 10 faces numérotées de 1 à 10. Au résultat, il ajoute sa force et soustrait l'endurance de l'adversaire. S'il est positif, le résultat est le nombre de points de vie perdus par l'adversaire.
Le but de cet exercice est de simuler le combat entre ces deux personnages.

1. Implémenter les deux personnages par des p-uplets nommés.

2. Créer la fonction `combat(perso1, perso2)` qui renvoie le nom du vainqueur ainsi que son nombre de points de vie restant. *On pourra utiliser la bibliothèque* `random`.