Programmation Python : Notions avancées

Cheat Sheet

Ensemble

`set()`	nouvel ensemble vide
`{1, 2, 3}`	nouvel ensemble avec des valeurs
`set(L)`	nouvel ensemble à partir d'une séquence `L`
`{x for x in E if` *cond*`}`	ensemble des `x` dans `E` qui satisfont *cond*
`len(s)`	taille de `s`
`v in s`	vaut `True` si `v` est dans `s`, `False` sinon
`for e in s:`	exécute le *bloc de code*
bloc de code	pour chaque élément `e` de `s`
`s.add(v)`	Ajoute la valeur `v` dans `s`
`s.remove(v)`	Retire la valeur `v` de `s`
`A - B`	différence entre `A` et `B`
`A | B`	union entre `A` et `B`
`A & B`	intersection entre `A` et `B`
`frozenset(L)`	ensemble non modifiable à partir de `L`

Dictionnaire

`{}`	nouveau dictionnaire vide
`{'a': 1, 'b': 2, 'c': 3}`	nouveau dictionnaire avec des valeurs
`d[k]`	valeur associée à la clé `k` dans `d`
`del(d[k])`	supprime la paire dont la clé est `k` de `d`
`k in d`	vaut `True` si `k` est une clé de `d`, `False` sinon
`for k in d:`	exécute le *bloc de code*
bloc de code	pour chaque clé `k` de `d`

Tuple

`()`	nouveau tuple vide
`(1, 2, 3)`	nouveau tuple avec des valeurs
`t[i]`	élément d'indice `i` de `t` (premier indice : 0)
`t[i:j]`	sous-tuple de `i` (inclus) à `j` (exclu)
`v in t`	vaut `True` si `v` est dans `t`, `False` sinon
`for e in t:`	exécute le *bloc de code*
bloc de code	pour chaque élément `e` de `t`
`t = 1, 2, 3`	emballage de valeurs dans un tuple
`a, b, c = t`	déballage d'un tuple dans des variables
`x, y = y, x`	échange des valeurs des variables `x` et `y`

Programmation orientée objet

`class` *Nom*`:`	définit une classe *Nom*
`def __init__(self,` p_1`, ..., `p_r`)`	définit un constructeur avec les r
bloc de code	paramètres positionnels p_1, ..., p_r
`self.var =` *val*	crée une variable d'instance `var`
`def m(self,` p_1`, ..., `p_s`)`	définit une méthode m avec les s
bloc de code	paramètres positionnels p_1, ..., p_s
`o = Nom(`v_1`, ..., `v_r`)`	crée une instance de la classe *Nom*
`o.var`	accède la variable d'instance `var` de `o`
`o.m(`w_1`, ..., `w_r`)`	appelle la méthode `m` sur `o`

Gestion d'erreur

`assert` *cond*	provoque une erreur si *cond* vaut `False`
`try:`	
1er bloc de code	exécute le *1er bloc de code*
`except:`	et arrête son exécution en cas d'erreur
2e bloc de code	pour exécuter le *2e bloc de code*
`finally:`	et dans tous les cas
3e bloc de code	exécute le *3er bloc de code*
`raise AnException()`	génère une erreur de type `AnException`
`except e:`	récupération de l'exception dans `e`
`except AnException:`	capture de l'erreur de type `AnException`
`except AnException as e:`	capture de l'erreur de type `AnException` dans `e`

Manipulation de fichier

`with open(`*path*`, 'r') as f:`	ouvre le fichier *path* en lecture seule
bloc de code	et exécute le *bloc de code*
`f.read()`	lit tout le fichier `f`
`f.readlines()`	lit tout le fichier `f` comme une liste de lignes
`for l in f:`	exécute le *bloc de code*
bloc de code	pour chaque ligne `l` de `f`
`f.write(s)`	écrit `s` dans le fichier `f`