Fiche 20 minutes

Types de données en python :

Nom du type	Exemples de valeur
Nombre	40 / 32.5 / -2
Chaîne de caractères	‘bonjour’ / ‘a’ / ‘’ / ‘plusieurs mots’ / ‘UAG’
Booléen	True / False
Liste	[1, 2, 3, 4] / [‘a’, ‘c’, ‘e’] / [] / [40, ‘a’, "bonjour"] / [[0, 1, 2], [3, 4, 5]]
Dictionnaire	{‘nom’ : ‘James’, ‘prénom’ : ‘Bond’} / {} / {True : ‘bonjour’, ‘abc’ : 40}
Type none	None

Pour exécuter un code Python (vous-même, sans ordinateur), il faut toujours lire et appliquer le code ligne par ligne, l’une après l’autre.

Conventions :

- On compte à partir de zéro
- Borne à gauche est inclue, borne à droite est exclue

Slicing :

S’applique aux listes. On parle de slicing quand on veut une plage d’éléments dans une liste ou une chaîne de caractères.

Examples :

```
mot = ‘bonjour’
mot[1] # vaut ‘o’
mot[-1] # vaut ‘r’
mot [2:4] # vaut ‘nj’
mot [:4] # vaut ‘bonj’
mot[2:] # vaut ‘njour’
```

Briques de bases du langage :

Variables :

```
age = 22
print(age)
```

Affecte la valeur 22 à la variable age, puis affiche le contenu de la variable age (22).

Structures conditionnelles :

```
age = ???
if age < 3:
      print(‘bébé’)
elif age < 13:
      print(‘enfant’)
elif age < 18:
      print(‘ado’)
else:
      print(‘adulte’)
```

Pour lire un code qui contient des structures conditionnelles (if/elif/else) on lit d’abord le if. Si la

condition du if est vraie, on exécute le code du bloc en dessous du if puis on passe tout le code jusqu'à la fin des éventuels elif/else.

On exécute le code d'un bloc elif si sa condition est vraie, et que la condition du if et des elif au-dessus ne le sont pas.

On exécute le code d'un else si aucune des conditions des if et elif ne sont vraies.

Ici, si age vaut 1 le code affiche « bébé », si age vaut 16 le code affiche « ado » (et seulement ado), etc…

Boucles :

La boucle while s'écrit :

```
while condition:
    ...
```

Le while se comporte exactement comme un if, sauf qu'à la fin du bloc de code (si on l'a exécuté), on remonte au niveau du while au lieu d'exécuter la suite du code. On reteste donc la condition. On ne peut sortir de cette « boucle » qu'une fois que la condition n'est pas/plus vraie.

La boucle for s'écrit :

```
for élément in liste:
    ...
```

On va ici exécuter le bloc de code une fois pour chaque élément dans la liste. élément contient donc successivement chaque élément de la liste.

Note : la fonction range génère une liste. Vous aurez ainsi range(4) qui vous génèrera [0, 1, 2, 3] ; range(2, 5) génèrera [2, 3, 4] ; range(0, 15, 3) génèrera [0, 3, 6, 9, 12]

Opérateur in :

Attention, à ne pas confondre avec le in utilisé avec le for.

```
élément in liste
clé in dictionnaire
```

élément in liste vaut True si l'élément est dans la liste et False sinon. Par exemple, 1 in [0, 1, 2] vaut True et 'bonjour' in [0, 1, 2] vaut False.

Pour les dictionnaires, 'nom' in {'nom' : 'James', 'prénom' : 'Bond'} vaut True et 'bar' in {'foo' : 'bar'} vaut False (ici bar n'est pas une clé du dico mais une de ses valeurs. 'bar' in {'foo' : 'bar'}.values() sera True).

Fonctions :

Définir une fonction est le fait de donner un nom à un bloc de code.

```
def is_even(n):
    print('check if', n, 'is even or not')
    return n % 2 == 0
```

On peut ensuite appeler la fonction : exécuter le bout de code :

```
print(is_even(2))
```

affichera :

```
check if 2 is even or not
True
```

Les fonctions peuvent prendre des paramètres (ici un paramètre n) et retourner une valeur.