Recapitulatif des syntaxes “de base” en Python

Demander et afficher des informations

Syntaxe	Description
`print("message")`	Affiche “message” dans la console
`v = input("message")`	Demande une valeur et la stocke dans `v`

Calculs

Syntaxe	Description
`a + b`	Addition de a b
`a - b`	Soustraction de a et b
`a / b`	Division de a par b
`a * b`	Multiplication de a par b
`a % b`	Modulo (reste de division) de a par b
`a ** b`	Exponentiation de a par b

Toutes ces opérations peuvent être appliquées directement sur une variable via la syntaxe du type `a += b` (additionner b à a et directement modifier la valeur de a avec le résultat).

Types de variable et conversion

Syntaxe	Description
`type(v)`	Renvoie le type de `v`
`int(v)`	Converti `v` en entier
`float(v)`	Converti `v` en float
`str(v)`	Converti `v` en string

Chaînes de caractères

Syntaxe	Description
`chaine1 + chaine2`	Concatène les chaînes de caractères `chaine1` et `chaine2`
`chaine[n:m]`	Retourne les caractères de `chaine` depuis la position n à m
`chaine * n`	Retourne `chaine` concaténée `n` fois avec elle-meme

Syntaxe	Description
`len(chaine)`	Retourne la longueur de `chaine`
`chaine.replace(a, b)`	Renvoie `chaine` avec les occurences de `a` remplacées par `b`
`chaine.split(c)`	Créé une liste à partir de `chaine` en la séparant par rapport au caractère `c`
`chaine.strip()`	"Nettoie" `chaine` en supprimant les espaces et `\n` au début et à la fin
`\n`	Représentation du caractère 'nouvelle ligne'

Fonctions

```
def ma_fonction(toto, tutu=3):
    une_valeur = toto * 6 + tutu
    return une_valeur
```

Cette fonction : - a pour nom `ma_fonction` ; - a pour argument `toto` et `tutu` ; - `tutu` est un argument optionnel avec comme valeur par défaut l'entier 3 ; - `une_valeur` est une variable locale à la fonction ; - elle retourne `une_valeur` ;

Conditions

```
if condition:
    instruction1
    instruction2
elif autre_condition:
    instruction3
elif encore_une_autre_condition:
    instruction4
else:
    instruction5
    instruction6
```

Opérateurs de conditions

Syntaxe	Description
`a == b`	Egalité entre `a` et `b`
`a != b`	Différence entre `a` et `b`
`a >  b`	`a` supérieur (strictement) à `b`
`a >= b`	`a` supérieur ou égal à `b`
`a <  b`	`a` inférieur (strictement) à `b`
`a <= b`	`a` inférieur ou égal à `b`

Syntaxe	Description
`cond1 and cond2`	`cond1` et `cond2`
`cond1 or cond2`	`cond1` ou `cond2`
`not cond`	négation de la condition `cond`
`a in b`	`a` est dans `b` (chaîne, liste, set..)

Inline `ifs`

```
parite = "pair" if n % 2 == 0 else "impair"
```

Exception, assertions

`try`/`except` permettent de tenter des instructions et d'attraper les exceptions qui peuvent survenir pour ensuite les gérer de manière spécifique :

```
try:
   instruction1
   instruction2
except FirstExceptionTime:
   instruction3
except Exception as e:
   print("an unknown exception happened ! :" + e.str)
```

Les assertions permettent d'expliciter et de vérifier des suppositions faites dans le code :

```
# Cette fonction fonctionne seulement pour des entiers premiers !
def une_fonction(n):
   assert isinstance(n, int) and is_prime(n)
```

Boucles

Syntaxe	Description
`for i in range(0, 10)`	Itère sur `i` de 0 à 9
`for element in stuff`	Itère sur tous les elements de stuff (liste, set, dictionnaire)
`for key, value in d.items()`	Itère sur toutes les clefs, valeurs du dictionnaire `d`
`while condition`	Répète un jeu d'instruction tant que `condition` est vraie
`break`	Quitte immédiatement une boucle

Syntaxe	Description
`continue`	Passe immédiatement à l'itération suivante d'une boucle

Structures de données

Syntaxe	Description
`L = ["a", 2, 3.14 ]`	Liste (suite ordonnée d'éléments)
`S = { "a", "b", 3 }`	Ensemble (éléments unique, désordonné)
`D = { "a": 2, "b": 4 }`	Dictionnaire (ensemble de clé-valeurs, avec clés uniques)
`T = (1,2,3)`	Tuple ou n-uplet (suite d'élément non-mutables)

Syntaxe	Description
`L[i]`	`i`-eme element d'une liste ou d'une tuple
`L[i:]`	Liste de tous les éléments à partir du `i`-eme
`L[i] = e`	Remplace le `i`-eme element par `e` dans une liste
`L.append(e)`	Ajoute `e` à la fin de la liste `L`
`S.add(e)`	Ajoute `e` dans le set `S`
`L.insert(i, e)`	Insère `e` à la position `i` dans la liste `L`
`chaine.join(L)`	Produit une string à partir de `L` en intercallant la string `chaine` entre les elements

Fichiers

Ouvrir et lire un fichier :

```
# Créé un contexte dans lequel le fichier
# est ouvert en lecture en tant que 'f',
# et met son contenu dans 'content'

with open("/un/fichier", "r") as f:
    content = f.readlines()
```

Ecrire dans un fichier :

```
# Créé un contexte dans lequel le fichier
```

```
# est ouvert en ré-écriture complète et
# écrit le contenu de 'content' dedans.

with open("/un/fichier", "w") as f:
    f.write(content)
```

(Le mode `'a'` (append) au lieu de `'w'` permet d'ouvrir le fichier pour ajouter du contenu à la fin plutôt que de le ré-écrire)