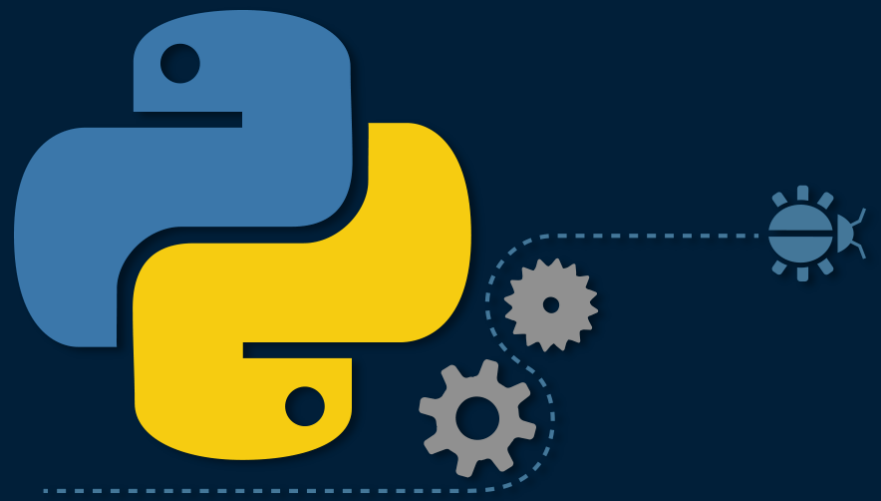


Programmation avec Python

Module 05 : Les variables complexes

OBJECTIFS

Savoir utiliser les énumérations

Savoir manipuler les chaînes de caractère ?

Savoir gérer des listes

Découvrir les dictionnaires

SOMMAIRE

- Définition :
  - Ensemble d'éléments non modifiables ordonnés
  - Liste de variables non dédoublonnées d'objets Python
  - Contient de zéro à plusieurs éléments
  - Il dispose d'une relation d'ordre et de méthodes

- Syntaxe : **(…)**
  - `tuple_explicite = tuple() # n-uplet vide explicite <class 'tuple'>`
  - `tuple_implicite = () # n-uplet vide implicite`
  - `print(a_tuple[index]) # accès à une valeur / indice de 0 à N-1`

- Exemples :
  - `t = (1) # type : <class 'int'> / parenthèses arithmétique !`
  - `t = (3, 4, 5,) # équivalent : t = 3, 4, 5`

```
tuple_days = (
    'Dimanche', # n°0
    'Lundi', # n°1
    'Mardi', # n°2
    'Mercredi', # n°3
    'Jeudi', # n°4
    'Vendredi', # n°5
    'Samedi', # n°6 (dernière virgule tolérée)
    )

print("Il y a ", len(tuple_days), "jours par semaine.")
print("Semaine de travail :")
print("du", tuple_days[1], "au", tuple_days[5]) # a_tuple[num_index]
```

Il y a 7 jours par semaine
Semaine de travail :
du Lundi au Vendredi

DÉMONSTRATION

Le n-uplet (tuple)

- Définition :
  - Ensemble d'éléments modifiables ordonnés
  - Liste de variables non dédoublonnées d'objets Python
  - Contient de zéro à plusieurs éléments
  - Il dispose d'une relation d'ordre et de méthodes

- Syntaxe : [...]
  - `list_explicit = list() # liste vide explicite <class 'list'>`
  - `list_implicit = [] # liste vide implicite`
  - `a_list[index] = "valeur" # affectation d'une valeur / indice de 0 à N-1`

- Exemples :
  - `lst = 1, 2, # type : <class 'tuple'> nb. élément = 2`
  - `lst = [1, 2,] # type : <class 'list'> nb. élément = 2`
  - `lst = [3, 4, "cinq",] # équivalent : lst = [3, 4, "cinq"]`
  - `lst[2] = 5 # redéfinition d'une valeur pour obtenir : lst = [3, 4, 5]`

- Appel d'une méthode (ou fonctionnalité) d'un objet :
  - *a_object.a_method(…)*
  - *a_list.method_of_list(…)*

- Principales fonctionnalités pour une liste :
  (accessible par variable)
  - `append, extend, insert, remove, …`
  - `count, pop, sort, reverse, …`

- Exemple :
  - Déplacer un élément dans la liste :

```
a_list.insert(2, a_list.pop(5)) # Déplacement de la position 6 à 3
```

```
list_days = [
    'Dimanche', # n°0
    'Lundi', # n°1
    'Mardi', # n°2
    'Mercredi', # n°3
    'Jeudi', # n°4
    'Vendredi', # n°5
    'Samedi', # n°6 (dernière virgule tolérée)
    ]
list_days.sort() # Triage des éléments dans la liste
for day in list_days:
    print(day)
 # Fin du bloc itératif
```

```
Dimanche
Jeudi
Lundi
Mardi
Mercredi
Samedi
Vendredi
```

```
list_days = [
    'Dimanche', # n°0
    'Lundi', # n°1
    'Mardi', # n°2
    'Mercredi', # n°3
    'Jeudi', # n°4
    'Vendredi', # n°5
    'Samedi', # n°6 (dernière virgule tolérée)
    ]
list_days.sort() # Triage des éléments dans la liste
for i, day in enumerate(list_days):
    print("n°", i, "=>", day)
 # Fin du bloc itératif
```

```
n° 0 => Dimanche
n° 1 => Jeudi
n° 2 => Lundi
n° 3 => Mardi
n° 4 => Mercredi
n° 5 => Samedi
n° 6 => Vendredi
```

- Il est possible de décrire un tableau à partir de plusieurs listes ;
  soit une liste contenant une autre liste par imbrication.
- Un tableau (ou matrice) peut donc avoir plusieurs dimensions
- Exemples de syntaxe :

```
a_tab_3x3 = [['A1', 'B1', 'C1'], ['A2', 'B2', 'C2'], ['A3', 'B3', 'C3']]
a_tab_4c = [['Col1', 'Col2', 'Col3', 'Col4']] # Ligne d'entête de colonne
a_tab_4c.append([11, 12, 13, 14]) # ligne n°2,
a_tab_4c.append([21, 22, 23, 24]) # ligne n°3 …
# Modifier la ligne n°3 / colonne n°5
a_tab[2][4] = "Nouvelle Valeur"
# Afficher la ligne n°8 / colonne n°4
print(a_tab[7][3])
```

DÉMONSTRATION

La liste

- Il existe plusieurs types de chaine de caractères :
  - `str # par défaut`
  - `unicode # encodage selon la norme unicode`
- Appel de méthodes d'une chaine de caractères :
  - `a_string.a_method_of_string(…)`
  - `"Une chaine de caractères".method_of_string(…).an_other_method(…)`
- Principales fonctionnalités sur une chaîne :
  (accessible par variable)
  - `lower, upper, capitalize, title, swapcase, …`
  - `len, count, index, find, replace, …`
  - `startswith, endswith, …`
  - `strip, split, format, …`
  - `isalpha, isdigit, …`

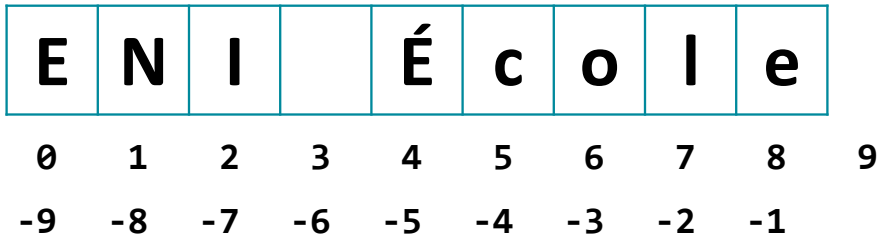
- Comparaison entre deux chaînes de caractères :
  - `string_A == string_B # Renvoi True ou False en fonction`
  - `a_string.lower() == a_string.upper() # Renverra False (sensible à la casse min/MAJ)`
- Découper des chaines de caractères :
  - `a_string[start=0:end=-1] # fin non incluse !`
- Exemples :
  - `msg = "Hello World"`
  - `msg[6] # Renvoi : 'W'`
  - `msg[0:4] # Renvoi : 'Hell'`
  - `msg[:5] # Renvoi : 'Hello'`
  - `msg[6:] # Renvoi : 'World'`
  - `msg[-3] # Renvoi : 'r'`
  - `msg[:-6] # Renvoi : 'Hello'`

```
# Parcourir une chaine de caractères :
sentence = "Hello World"
for letter in sentence:
    print(letter)
 # Fin du bloc itératif
```

```
H
e
l
l
o

W
o
r
l
d
```

- Le découpage de chaînes de caractères (slice) s'appuie sur un système d'indexes :



- Les indexes vont permettre d'indiquer les portions que l'on veut extraire.

- Il existe un 3ème paramètre permettant d'indiquer le pas :
  - `a_string[start=0:end=-1:step=1] # définit par défaut à 1`
- Exemples :
  - `quotation = "Seul on va plus vite, ensemble on va plus loin."`
  - `quotation[::2] # 1/2 lettre : Su nv lsvt,esml nv lsli.`
  - `quotation[::3] # 1/3 lettre : Slnal t sb   uln`

- Il est possible de construire une chaîne de caractères formatée :
  - via l'opérateur *modulo* **%** :
    - `"Tu t'inspires de %s ?"%"C"  # Tu t'inspires de C ?`
    - `"Choisis la %s %s, et tout s'arrête."  % ("pilule", "bleue")`
    - `"Après tu %(verb) faire de beaux %(cpl), …"%{"cpl": "rêves", "verb": "pourras"} # … et de penser ce que tu veux.`
  - via la méthode **format()** :
    - `"Tu ne t'inspirerais pas du {0} ?".format("C++")  # Tu ne t'inspirerais pas du C++ ?`
    - `"Choisis la {0} {1} :".format("pilule", "rouge") # Choisis la pilule rouge :`
    - `"tu restes au {} des {}, …".format("Pays", "Merveilles")`
    - `"et on {verb} avec le {ani} …".format(ani="lapin blanc", verb="descend") # … au fond du gouffre.`

- Il est également possible de formater un numérique pour l'affichage :
  - via l'opérateur *modulo* **%** : `decimal = 354.786`
    - `"Un total de %i €" % decimal # Un total de 354 € (valeur tronquée)`
    - `"Un total de %f €" % decimal # Un total de 354.786000 € (6 décimales par défaut)`
    - `"Un total de %.0f €" % decimal # Un total de 355 € (avec arrondi sup.)`
    - `"Un total de %.2f €" % decimal # Un total de 354.79 € (avec arrondi sup.)`
  - via la méthode **format()** :
    - `"Référence = {0:06}".format(123)  # Référence = 000123 (zéro significatif)`
    - `"CP = {0:05}, PIN = {1:04}".format(1400, 654)  # CP : 01400, PIN = 0654`
    - `"Un total de {:.2f} €".format(234.56987) # Un total de 234.57 €`
    - `"Un total de {:09.3f} €".format(652.45291) # Un total de 00652.453 €`
    - …

- Les chaines de caractères formatées (aussi appelées f-strings) permettent d'inclure dans celle-ci la valeur d'une variable ou bien une expression.
  - Préfixer une chaine de caractère par **f** ou **F** :
    - ```
      a_variable = "la maison"
      f"Bienvenue à {a_variable} !"  # Bienvenue à la maison !
      ```
    - ```
      import math
      precision = 4
      print(F"La valeur approximative de pi à {precision} décimales est de "
            F"{math.pi:.{precision}f}!")
      # soit {math.pi:.4f}
      # La valeur approximative de pi à 4 décimales est de 3.1416 !
      ```

- Liste des principaux caractères spéciaux :
  - `\'` apostrophe
  - `\"` guillemet
  - `\\` anti-slash
  - `\t` tabulation
  - `\b` retour arrière (d'un caractère)
  - `\r` retour en début de ligne
  - `\n` passage à la ligne suivante
- Exemple :

```
print("J\'apprécie un message\nsur \"plusieurs\" lignes\navec\tune tabulation !")
```

```
J'apprécie un message
sur "plusieurs" lignes
avec    une tabulation !
```

DÉMONSTRATION

Les chaînes de caractères

OPÉRATEUR TERNAIRE

Équivalent / Conditionnelle	Exemples d'Alternative Syntaxique

`def iif(condition, true_val, false_val):`
`  if condition:`
`  return true_val`
`  else:`
`  return false_val`
	`true_val if condition else false_val`
	
	`choice_1 if cond_1 else choice_2 if cond_2 else choice_3`

	`(false_val, true_val)[condition]`
	
	`(choice_3, (choice_2, choice_1)[cond_1])[cond_2]`

`def what_else(condition, else_val):`
`  if condition:`
`  return condition`
`  else:`
`  return else_val`
	`condition or else_val`
	
	`predicat_1 or predicat_2 or else_val`

- Définition : {*key*:*value*…}
  - Ensemble d'éléments modifiables ordonnés
  - Indexé à partir de clés uniques (de types distincts)
  - Contient de zéro à plusieurs éléments
  - Il dispose d'une relation d'ordre d'insertion et de méthodes spécifiques
- Syntaxe :
  - `dict_explicit = dict() # dictionnaire vide explicite <class 'dict'>`
  - `dict_implicit = {} # dictionnaire vide implicite`
- Exemples :
  - `dict_type = {"entier":2, "décimal":37.4, "chaîne":"de caractères"} # clef:valeur`
  - `dict_cooking = {"oeuf":3, "farine":250, 180:"°c"}`
  - `dict_cooking['beurre'] = 80 # ajout d'un nouvel élément (à la fin)`
  - `print(dict_cooking["beurre"]) # affiche 80`

```python
some_fruit = {"Pommes":8, "Poires":5, "Kiwi":12}
print(some_fruit)

# parcourir les clés
for key in some_fruit: # équivalent à 'for key in some_fruit.keys():'
    print(key)

# parcourir les valeurs
for valeur in some_fruit.values():
    print(valeur)

# parcourir les clés avec valeurs
print("\nStock de fruits :")
for key, value in some_fruit.items():
    print(" - {}x {}".format(value, key))
```

```
{'Pommes': 8, 'Poires': 5, 'Kiwi': 12}
Pommes
Poires
Kiwi
8
5
12

Stock de fruits :
 - 8x Pommes
 - 5x Poires
 - 12x Kiwi
```

DÉMONSTRATION

Les syntaxes avancées

TRAVAUX PRATIQUES

Manipuler des variables complexes