INF1221 : PROGRAMMATION PYTHON

AMOUZOU Grâce Dorcas Akpéné / EPL / 2024-2025 M

BEFOR WE START

Démarrez l'interpréteur Python :

- ★ dans l’invite de commandes
- ★ dans Pycharm
- ★ dans Spyder

SÉANCE 3 - PARTIE 1: APPROFONDISSEMENT

ECRITURE FORMATÉE AVEC F-STRINGS 1/3

Utilisation des f-strings

- x = 32
- nom = "John"
- print(f"{nom} a {x} ans")

- var = "to"
- print(f"{var} et {var} font {var}{var}")

ECRITURE FORMATÉE AVEC F-STRINGS 2/3

Les f-strings permettent de remplacer des variables au sein d'une chaîne de caractères. On peut également spécifier le format de leur affichage.

- ★ var = 1/3
- ★ print("1 divisé par 3 donne", var)

- ★ var = 1/3
- ★ print(f"1 divisé par 3 donne {var:.2f}")

ECRITURE FORMATÉE AVEC F-STRINGS 3/3

- ★ Notez que > spécifie un alignement à droite, < spécifie un alignement à gauche et ^ spécifie un alignement centré.
- ★ Il est également possible d'indiquer le caractère qui servira de remplissage lors des alignements (l'espace est le caractère par défaut).
- ★ La lettre d (ici d veut dire decimal integer). Ce formatage est également possible sur des chaînes de caractères avec la lettre s (comme string)

- ★ print(f"{10:*^6d}") ; print(f"{1000:*^6d}")

>>**10**

>> *1000*

EXERCICE

Trouvez comment réaliser l'affichage suivant :

- *****1******
- *****1*****
- ****1*****
- ****1****
- ***1****
- ***1***
- **1***
- **1**
- *1**
- *1*
- 1*
- 1

FORMATAGE 1/3

Formatage des chaînes de caractères

★ print("atom HN") ; print("atom HDE1")

>> atom HN
>> atom HDE1

★ print(f"atom {'HN':>4s}") ; print(f"atom {'HDE1':>4s}")

>> atom HN
>> atom HDE1

★ print(f"1 / 1000 = {var:100.5f}") ⇒ **Autre cas**

FORMATAGE 2/3

Formatage des chaînes de caractères

- ★ print(f"Accolades littérales {{}} ou {{ ou }}" f"et pour le formatage {10}")
- ★ print(f"accolades sans variable {}")

FORMATAGE 3/3

Une fonctionnalité extrêmement puissante des f-strings est de supporter des expressions Python au sein des accolades.

- ★ print(f"Le résultat de 5 * 5 vaut {5 * 5}")
- ★ print(f"Résultat d'une opération avec des floats : {(4.1 * 6.7)}")
- ★ print(f"Le minimum est {min(1, -2, 4)}")
- ★ entier = 2
- ★ print(f"Le type de {entier} est {type(entier)}")

EXERCICES 1/2

- ★ Générez en une ligne de code un brin d'ADN poly-A (AAAA…) de 20 bases suivi d'un poly-GC régulier (GCGCGC…) de 40 bases.
- ★ Dans un script percGC.py, calculez un pourcentage de GC avec l'instruction suivante : perc_GC = ((4500 + 2575) / 14800) * 100
- ★ Ensuite, affichez le contenu de la variable perc_GC à l'écran avec 0, 1, 2 puis 3 décimales sous forme arrondie en utilisant l'écriture formatée et les f-strings. On souhaite que le programme affiche la sortie suivante :

```
Le pourcentage de GC est 48     %
Le pourcentage de GC est 47.8   %
Le pourcentage de GC est 47.80  %
Le pourcentage de GC est 47.804 %
```

EXERCICES 2/2

- ★ Centrez le nombre 42 sur 10 caractères, avec des -.
- ★ Alignez 7 à droite sur 5 caractères, avec des 0.
- ★ Alignez 99 à gauche sur 6 caractères avec des #.
- ★ Formatez une phrase : “NOM” vit à “VILLE”. Nom et Ville sont entrés par l’utilisateur.
- ★ Affichez la table de multiplication d’un nombre entré par l’utilisateur.
- ★ Formatez PI sur 3 décimales (prendre PI dans la bibliothèque MATH).
- ★ BONUS : Affichez en hexadécimal un nombre entré par l’utilisateur.

SÉANCE 3 - PARTIE 2:
LES LISTES

DÉFINITION

- ★ Une liste est une structure de données qui contient une collection d'objets Python. Elle contient une séquence d'autres objets.
- ★ Python autorise la construction de liste contenant des valeurs de types différents (par exemple entier et chaîne de caractères)
- ★ Une liste est déclarée par une série de valeurs (n'oubliez pas les guillemets, simples ou doubles, s'il s'agit de chaînes de caractères) séparées par des virgules, et le tout encadré par des crochets.
- ★ Une liste vide : liste_vide = []

EXEMPLES

- ★ animaux = ["girafe", "tigre", "singe", "souris"]
- ★ animaux
- ★ tailles = [5, 2.5, 1.75, 0.15]
- ★ tailles
- ★ mixte = ["girafe", 5, "souris", 0.15]
- ★ mixte

UTILISATION

- ★ Un des gros avantages d'une liste est que vous accédez à ses éléments par leur position. Ce numéro est appelé indice (ou index) de la liste.
- ★

```
liste  : ["girafe", "tigre", "singe", "souris"]
indice :        0        1        2         3
```

- ★ animaux = ["girafe", "tigre", "singe", "souris"]
- ★ animaux[0]
- ★ animaux[1]
- ★ animaux[2]
- ★ animaux[3]

OPÉRATIONS SUR LES LISTES

- ★ L'opérateur + pour la concaténation
- ★ L'opérateur * pour la duplication
- ★ Pour ajouter un élément à la liste : utiliser la méthode .append() ou +

- ★ animaux = ["girafe", "tigre", "singe", "souris"]
- ★ animaux[0]
- ★ animaux[1]
- ★ animaux[2]
- ★ animaux[3]

INDIÇAGE NÉGATIF

La liste peut également être indexée avec des nombres négatifs selon le modèle suivant :

```
liste          : ["girafe", "tigre", "singe", "souris"]
indice positif :        0        1        2         3
indice négatif :       -4       -3       -2        -1
```

Les indices négatifs reviennent à compter à partir de la fin. Leur principal avantage est que vous pouvez accéder au

dernier élément d'une liste à l'aide de l'indice -1 sans pour autant connaître la longueur de cette liste. L'avant-dernier

élément a lui l'indice -2, l'avant-avant dernier l'indice -3, etc.

TRANCHES 1/2

Un autre avantage des listes est la possibilité de sélectionner une partie d'une liste en utilisant un indiçage construit sur le modèle [m:n+1] pour récupérer tous les éléments, du **émième** au **énième** (de l'élément **m inclu** à l'élément **n+1 exclu**). On dit alors qu'on récupère une **tranche** de la liste.

- ★ animaux = ["girafe", "tigre", "singe", "souris"]
- ★ animaux[0:2]
- ★ animaux[1:]
- ★ animaux[:2]
- ★ animaux[:]
- ★ animaux[1:-1]

TRANCHES 2/2

- ★ x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- ★ x
- ★ x[0:9:2]
- ★ x[::2]
- ★ x[::3]
- ★ x[1:6:3]
- ★ x[0:6:3]

AUTRES MÉTHODES

- ★ len(animaux)
- ★ len(x)
- ★ list(range(10))
- ★ list(range(15, 20))
- ★ list(range(0, 1000, 200))
- ★ list(range(2, -2, -1))
- ★ list(range(10, 0, -1))
- ★ sum(x)
- ★ min(x)
- ★ max(x)

LISTE DE LISTES

- ★ prairie1 = ["girafe", 4]
- ★ prairie2 = ["tigre", 2]
- ★ prairie3 = ["singe", 5]
- ★ savane = [prairie1, prairie2, prairie3]
- ★ savane
- ★ savane[1]
- ★ savane[1][0]
- ★ savane[1][1]

EXERCICES

Prédisez le comportement de chaque instruction ci-dessous :

```
liste1 = list(range(10, 15))
var = 0
var2 = 10
```

- ★ print(liste1[2])
- ★ print(liste1[var])
- ★ print(liste1[var2])
- ★ print(liste1["var"])

Lorsqu'une instruction produit une erreur, identifiez pourquoi.

SÉANCE 3 :
END