

Fonctions et chaînes de caractères

Initiation à l'algorithmique et à la programmation

Semaine 3 – INALCO

Revekka Kyriakoglou

Objectifs de la séance

- Définir une fonction et l'appeler avec des arguments.
- Distinguer afficher un résultat et renvoyer une valeur.
- Manipuler une chaîne de caractères.
- Combiner plusieurs fonctions pour traiter un texte.
- Tester un programme avec plusieurs entrées.

Rappel : variables et types

Avant d'exécuter le programme, prévoir ce qu'il affiche.

```
1 prenom = "Alice"
2 annee = 2026
3 print(prenom)
4 print(type(prenom))
5 print(str(annee) + " : " + prenom)
```

`str` est le type des chaînes. `str(annee)` convertit un entier en chaîne.

Rappel : une entrée est une chaîne

Quel résultat obtient-on si on saisit 12 ?

```
1 valeur = input("Un nombre : ")
2 print(valeur + valeur)
3 print(int(valeur) + int(valeur))
```

`input()` renvoie une chaîne. La saisie doit représenter un entier pour que `int()` fonctionne.

Pourquoi des fonctions ?

- Éviter de répéter le même code.
- Donner un nom à une opération.
- Décomposer un problème en étapes.
- Réutiliser et tester chaque étape séparément.

Exemple : nettoyer un texte avant d'en compter les caractères.

Une opération répétée

Nous calculons deux aires de rectangle.

```
1 longueur = 5
2 largeur = 3
3 print(longueur * largeur)
4
5 longueur = 8
6 largeur = 2
7 print(longueur * largeur)
```

Le calcul reste le même. Seules les données changent.

Définir une fonction

Une fonction regroupe des instructions sous un nom.

```
1 def calculer_aire(longueur, largeur):  
2     aire = longueur * largeur  
3     return aire
```

- `def` introduit la définition.
- Les paramètres reçoivent les données de l'appel.
- L'indentation délimite le corps de la fonction.
- `return` renvoie le résultat.

Définition et appel

La définition seule n'exécute pas le corps de la fonction.

```
1 def calculer_aire(longueur, largeur):  
2     return longueur * largeur  
3  
4 resultat = calculer_aire(5, 3)  
5 print(resultat)
```

Lors de l'appel, Python calcule $5 * 3$, renvoie 15, puis l'affecte à `resultat`.

Paramètres et arguments

Deux mots pour deux rôles.

```
1 def calculer_aire(longueur, largeur):  
2     return longueur * largeur  
3  
4 print(calculer_aire(5, 3))
```

- longueur et largeur : les **paramètres** de la définition.
- 5 et 3 : les **arguments** de cet appel.
- Les arguments correspondent ici aux paramètres dans l'ordre.

Réutiliser une fonction

Un seul calcul défini, plusieurs appels.

```
1 def calculer_aire(longueur, largeur):  
2     return longueur * largeur  
3  
4 print(calculer_aire(5, 3))  
5 print(calculer_aire(8, 2))  
6 print(calculer_aire(4, 4))
```

Prévoir les trois valeurs affichées.

Exercice 1 : une première fonction

Travail individuel, puis comparaison à deux (8 min).

- Écrire `doubler(nombre)` qui renvoie le double.
- Appeler cette fonction avec 4, 0 et -3.
- Stocker un résultat dans une variable, puis l'afficher.
- Avant chaque exécution, écrire la valeur attendue.

Afficher un message

Cette fonction effectue une action visible.

```
1 def dire_bonjour(prenom):  
2     print("Bonjour", prenom)  
3  
4 dire_bonjour("Alice")
```

Elle affiche un message. Sans `return` explicite, elle renvoie la valeur `None`.

Renvoyer un message

Cette fonction fournit une valeur que l'on peut réutiliser.

```
1 def creer_message(prenom) :  
2     return "Bonjour " + prenom  
3  
4 message = creer_message("Alice")  
5 print(message)  
6 print(len(message))
```

`len(message)` donne le nombre de caractères, espaces compris.

La différence entre print et return

Prévoir les deux lignes affichées.

```
1 def dire_bonjour(prenom) :  
2     print("Bonjour", prenom)  
3  
4 resultat = dire_bonjour("Alice")  
5 print(resultat)
```

Affichage : Bonjour Alice, puis None.

Le texte affiché n'est pas automatiquement la valeur renvoyée.

return termine l'appel

Que va afficher ce programme ?

```
1 def doubler(nombre):  
2     return nombre * 2  
3     print("Calcul termine")  
4  
5 print(doubler(4))
```

Python affiche 8. L'instruction après `return` n'est pas exécutée dans cet appel.

Exercice 2 : réutiliser une valeur

```
1 def ajouter_un(nombre):  
2     print(nombre + 1)  
3  
4 print(resultat)
```

- Prévoir l'affichage actuel.
- Modifier la fonction pour qu'elle **renvoie** le résultat.
- Afficher ensuite le double de cette valeur.

Les variables locales

La variable créée dans la fonction reste locale à cet appel.

```
1 def calculer_aire(longueur, largeur):  
2     aire = longueur * largeur  
3     return aire  
4  
5 resultat = calculer_aire(5, 3)  
6 print(resultat)  
7 print(aire)
```

La dernière ligne provoque une `NameError` : aucune variable `aire` n'a été définie en dehors de la fonction.

Des variables de même nom

Prévoir l'affichage.

```
1 nombre = 10
2
3 def doubler(nombre):
4     nombre = nombre * 2
5     return nombre
6
7 print(doubler(3))
8 print(nombre)
```

Résultats : 6, puis 10. Le paramètre `nombre` est local à la fonction.

Documenter une fonction

Une docstring décrit ce que fait la fonction.

```
1 def calculer_aire(longueur, largeur):  
2     """Renvoie l'aire d'un rectangle."""  
3     return longueur * largeur
```

- Placer la docstring au début du corps.
- Décrire le résultat avec une phrase courte.
- Choisir un nom qui précise l'opération.

Tester une fonction

Pour chaque appel, comparer le résultat à la valeur attendue.

```
1 def calculer_aire(longueur, largeur):  
2     return longueur * largeur  
3  
4 print(calculer_aire(5, 3))    # attendu : 15  
5 print(calculer_aire(4, 4))    # attendu : 16  
6 print(calculer_aire(0, 7))    # attendu : 0
```

Ici, les longueurs prévues sont des nombres positifs ou nuls.
Les tests vérifient des cas ordinaires et un cas limite.

Le type str

Une chaîne de caractères représente du texte.

```
1 mot = "bonjour"
2 phrase = 'Je parle français.'
3 vide = ""
4
5 print(type(mot))
6 print(len(vide))
```

- On peut utiliser des guillemets simples ou doubles.
- Une chaîne peut contenir des espaces et de la ponctuation.
- Une chaîne vide contient zéro caractère.

La longueur d'une chaîne

La fonction intégrée `len` compte les caractères.

```
1 print(len("chat"))  
2 print(len("un chat"))  
3 print(len("bonjour!"))  
4 print(len(""))
```

Résultats : 4, 7, 8, 0.

Les espaces et la ponctuation comptent aussi.

On utilise ici des textes simples, sans caractères combinés ni emoji.

La concaténation

L'opérateur + assemble deux chaînes.

```
1 prenom = "Alice"
2 nom = "Martin"
3 nom_complet = prenom + " " + nom
4 print(nom_complet)
5
6 print("Cours " + str(3))
```

Un espace doit être ajouté explicitement.

Pour concaténer un nombre avec du texte, on peut le convertir avec `str()`.

Fonction et méthode

Deux syntaxes utiles pour travailler avec du texte.

```
1 texte = "Bonjour"
2
3 print(len(texte))
4 print(texte.lower())
```

- `len(texte)` : appel d'une fonction.
- `texte.lower()` : appel d'une méthode de la chaîne.
- Le point et les parenthèses font partie de la syntaxe.

Minuscules et majuscules

Les méthodes `lower` et `upper` renvoient une nouvelle chaîne.

```
1 texte = "Bonjour Paris"
2
3 print(texte.lower())
4 print(texte.upper())
5 print(texte)
```

Résultats : `bonjour paris`, `BONJOUR PARIS`, puis `Bonjour Paris`.

Les espaces aux extrémités

La méthode `strip` retire les espaces blancs au début et à la fin.

```
1 texte = "  un chat  "
2 nettoye = texte.strip()
3
4 print("[ " + nettoye + " ]")
5 print(len(nettoye))
```

Résultats : `[un chat]` et 7.

L'espace entre `un` et `chat` reste présent.

Remplacer une séquence

La méthode `replace` renvoie une chaîne avec les remplacements.

```
1 phrase = "un chat et un chaton"
2 print(phrase.replace("chat", "chien"))
3
4 print("un petit chat".replace(" ", ""))
```

Résultats : un chien et un chienon, puis unpetitchat.
`replace()` remplace une séquence de caractères, même à l'intérieur d'un mot.

Les chaînes sont immuables

Une méthode de transformation produit une nouvelle chaîne.

```
1 texte = "BONJOUR"  
2 texte.lower()  
3 print(texte)  
4  
5 texte = texte.lower()  
6 print(texte)
```

Résultats : BONJOUR, puis bonjour.

L'affectation associe ensuite le nom `texte` au nouveau résultat.

Exercice 3 : prévoir les résultats

Pour chaque ligne, noter la valeur affichée.

```
1 texte = "  Bonjour Paris  "  
2 print(len(texte))  
3 print(texte.strip())  
4 print(texte.lower())  
5 print(texte.replace("Paris", "Lyon"))  
6 print(texte)
```

Dans quelles lignes les espaces aux extrémités sont-ils encore présents ?

Accéder à un caractère

Les indices commencent à zéro.

```
1 mot = "chat"  
2 print(mot[0])  
3 print(mot[1])  
4 print(mot[3])  
5 print(mot[-1])
```

Caractère	c	h	a	t
Indice	0	1	2	3

Un indice négatif compte à partir de la fin.

Les limites des indices

Quel indice dépasse la longueur du mot ?

```
1 mot = "chat"  
2 print(mot[4])
```

Cette ligne provoque une `IndexError`.

Pour une chaîne de longueur 4, les indices positifs valides vont de 0 à 3.

La chaîne vide ne possède aucun caractère accessible.

Extraire une partie du texte

Syntaxe : `texte[debut:fin]`. La borne fin est exclue.

```
1 mot = "bonjour"
2 print (mot [0:3])
3 print (mot [:3])
4 print (mot [3:])
5 print (mot [-2:])
```

Résultats : bon, bon, jour, ur.

Une borne omise correspond au début ou à la fin de la chaîne.

Exercice 4 : indices et tranches

Prévoir, puis vérifier (5 min).

```
1 mot = "langue"  
2 print (mot [0])  
3 print (mot [-1])  
4 print (mot [1:4])  
5 print (mot [:2])  
6 print (mot [2:])
```

Écrire aussi une expression qui extrait les trois derniers caractères.

Une fonction pour nettoyer un texte

Combiner deux méthodes déjà connues.

```
1 def nettoyer_texte(texte):  
2     """Retire les espaces externes et met en  
   minuscules."""  
3     sans_espaces = texte.strip()  
4     return sans_espaces.lower()  
5  
6 print(nettoyer_texte("  Bonjour  "))
```

Résultat : bonjour. Les espaces internes restent présents.

Enchaîner des méthodes

Une écriture plus courte pour la même opération.

```
1 def nettoyer_texte(texte):  
2     """Retire les espaces externes et met en  
   minuscules."""  
3     return texte.strip().lower()
```

Python calcule d'abord `texte.strip()`, puis applique `lower()` à la chaîne obtenue.

On peut garder des étapes séparées si elles facilitent la lecture.

Exercice 5 : formater un nom

- Écrire `formater_nom(prenom, nom)`.
- Retirer les espaces aux extrémités de chaque argument.
- Mettre le nom de famille en majuscules.
- Renvoyer le prénom et le nom séparés par un espace.

Exemple : (`" Alice "`, `" martin "`) doit donner `"Alice MARTIN"`.

Une fonction qui appelle une autre fonction

Le résultat du nettoyage devient l'entrée du comptage.

```
1 def longueur_texte_nettoye(texte):  
2     """Compte les caracteres apres nettoyage."""  
3     texte_nettoye = nettoyer_texte(texte)  
4     return len(texte_nettoye)  
5  
6 print(longueur_texte_nettoye("  Un chat  "))
```

Résultat : 7. Définir aussi `nettoyer_texte` dans le fichier avant cet appel.

Exercice 6 : compter sans les espaces

- Écrire `compter_sans_espaces(texte)`.
- Retirer les espaces ordinaires avec une méthode connue.
- Renvoyer la longueur de la chaîne obtenue.
- Tester avec `"un chat"`, `" "` et `""`.

Valeurs attendues : 6, 0, 0.

Ici, on retire uniquement le caractère espace `" "`.

Petit programme : une fiche de texte

Travail individuel (15 min, fin incluse dans le bilan). Créer un fichier contenant les fonctions précédentes, puis :

- 1 Demander une phrase avec `input()`.
- 2 Obtenir la phrase nettoyée avec `nettoyer_texte()`.
- 3 Afficher cette phrase et sa longueur.
- 4 Afficher son nombre de caractères sans espaces.
- 5 Afficher ses trois premiers caractères.

Tester une phrase ordinaire, une phrase avec des espaces aux extrémités et une saisie vide.

Méthode de travail

- 1 Préciser les entrées et le résultat attendu.
- 2 Écrire une petite fonction pour chaque opération.
- 3 Prévoir le résultat sur un exemple simple.
- 4 Exécuter et comparer.
- 5 Lire le message d'erreur si nécessaire.
- 6 Tester aussi une entrée vide ou une valeur limite.

Vérification des acquis

Sans recopier un exemple (5 min).

- Écrire `trois_preiers(texte)` qui renvoie les trois premiers caractères après `strip()`.
- Que renvoie-t-elle pour une chaîne vide ?
- Expliquer en une phrase la différence entre paramètre et argument.
- Expliquer la différence entre `print()` et `return`.

À retenir

- `def` définit une fonction. L'appel exécute son corps.
- Les paramètres reçoivent les arguments de l'appel.
- `return` renvoie une valeur et termine l'appel.
- `print()` affiche une information.
- Les méthodes de transformation renvoient de nouvelles chaînes.
- Les indices commencent à zéro. La fin d'une tranche est exclue.
- Tester chaque fonction sur plusieurs entrées.