



ISN – Informatique et Sciences du Numérique

TP1 PYTHON : AFFICHER DES CARACTERES

1 – INSTRUCTION « PRINT »

L'instruction « `print` » permet d'afficher des résultats de calculs ou des caractères.

Exercice 1

1. **Taper**, dans l'interpréteur, les 2 lignes de l'exemple ci-dessous.

```
>>> print("J'apprends à programmer en Python")
>>> print(2 + 2)
```

2. **Taper** à nouveau ces 2 lignes dans l'éditeur de scripts. **Enregistrer** puis **Exécuter** le script.

2 – AFFICHER UNE CHAÎNE DE CARACTERES

Les chaînes de caractères sont encadrées par des guillemets ou des apostrophes.

Exercice 2

1. **Taper**, dans l'interpréteur, les lignes suivantes :

```
>>> print('Il apprend à programmer en Python')
>>> print("Il dit : 'J'apprends à programmer en Python'")
>>> print('Il dit : "J'apprends à programmer en Python"')
>>> print("Il dit : J'apprends à programmer en Python")
```

2. **Justifier** pourquoi certaines lignes provoquent des erreurs.

Le caractère « `\` » permet d'indiquer à l'interpréteur que le caractère qui le suit doit être converti tel quel et ne doit pas être interprété.



Exercice 3

1. **Taper**, dans l'interpréteur, les lignes suivantes :

```
>>> print("Il dit : \"J'apprends à programmer en Python\"")
>>> print('Il dit : "J'apprends à programmer en Python"')
```

2. **Justifier** pourquoi ces lignes ne provoquent plus d'erreurs.

Pour afficher une chaîne de caractères sur plusieurs lignes il faut utiliser les caractères « \n » (retour à la ligne).

Exercice 4

Taper, dans l'interpréteur, la ligne suivante :

```
>>> print("#####\n#      #      #\n#####  #")
```

Il est également possible d'utiliser des triples guillemets ou des triples apostrophes pour ouvrir et fermer la chaîne de caractères.

Exercice 5

Taper, dans l'interpréteur, les lignes suivantes :

```
>>> print("""#####
...#      #      #
...###      #####  #""")
```

3 – STOCKAGE D'UNE CHAÎNE DE CARACTÈRES DANS UNE VARIABLE

*Comme tout type de donnée, une chaîne de caractères peut être **stockée dans une variable**.*

Exercice 6

1. **Taper**, dans l'interpréteur, les lignes suivantes :

```
>>> ma_variable = "J'apprends à programmer en Python"
>>> print(ma_variable)
```

2. **Taper**, dans l'interpréteur, les lignes suivantes :

```
>>> niveau_1 = """
...#####
...#      #      #
...###      #####  #"""
>>> print(niveau_1)
```

3. **Proposer** une solution permettant d'éviter que le premier saut à la ligne, après le premier triple guillemet soit conservé et ainsi éviter l'affichage d'une ligne vide.



4 – OPERATIONS SUR LES CHAINES DE CARACTERES

La **multiplication** de chaînes de caractères est réalisée à l'aide de l'opérateur « * ». La **concaténation** de chaînes de caractères est réalisée à l'aide de l'opérateur « + ».

Exercice 7

1. **Taper**, dans l'interpréteur, la ligne suivante :

```
>>> print("Coucou " * 3)
```

2. **Taper**, dans l'interpréteur, les lignes suivantes :

```
>>> mur_complet = "#"  
>>> print(mur_complet * 20 )
```

3. **Taper**, dans l'interpréteur, les lignes suivantes :

```
>>> chaine= "Bonjour, " + "C'est " + "moi."  
>>> print(chaine)
```

Il n'est pas possible de concaténer une chaîne de caractères avec un nombre. Pour convertir un nombre en chaîne de caractère il faut utiliser l'instruction « str ».

Exercice 8

1. **Taper**, dans l'interpréteur, la ligne suivante :

```
>>> print("Score :" + 35)
```

2. **Justifier** pourquoi cette ligne provoque une erreur.

3. **Taper**, dans l'interpréteur, la ligne suivante :

```
>>> print("Score :" + str(35))
```

5 – LISTES DE CARACTERES

Une **liste** est une **structure de données** pouvant contenir des éléments de différents types (chaîne de caractères, entiers...) organisés sous la **forme d'un tableau** et accessibles grâce à un **index**.

Pour définir une liste, il suffit d'indiquer les différents éléments entre crochets.

Pour accéder aux différents éléments il faut indiquer entre crochet le numéro d'index (de 0 à n-1 pour une liste de n éléments).



Exercice 9

Taper les lignes suivantes. **Justifier** l’affichage obtenu. Indiquer pourquoi la dernière ligne provoque une erreur.

```
>>>niveau_1 = ["#####", "# # # #", "### ##### #"]
>>>print(niveau_1[0])
>>>print(niveau_1[0][1])
>>>print(niveau_1[1][1])
>>>print(niveau_1[3][1])
```

6 – UTILISATION DE BOUCLES POUR AFFICHER DES LISTES

Pour afficher une liste, il faut utiliser l’instruction « `print` » pour chacun des éléments. Pour éviter d’avoir à répéter plusieurs fois l’instruction « `print` ».

La boucle « `for` » est parfaitement adaptée au parcours d’éléments d’une liste. La syntaxe est la suivante : « `for` `element` `in` `liste` ». Pour chaque élément de la liste le programme réalisera l’instruction qui suit la ligne « `for` ».

Exercice 10

1. **Taper** dans l’éditeur de scripts les lignes suivantes :

```
niveau_1 = ["#####", "# # # #", "### ##### #"]
for ligne in niveau_1 :
    print(ligne)
```

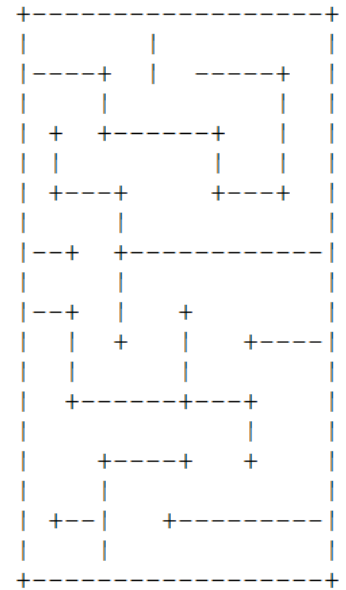
2. **Enregistrer** puis **Exécuter** le script.



7 – LABYRINTHE

Le premier labyrinthe est réalisé à l'aide d'une liste contenant **20 chaînes de caractères** composées de **20 caractères**. Le caractère « - » sera utilisé pour réaliser les **murs horizontaux**. Les **murs verticaux** seront réalisés à l'aide du caractère « | ». Les **angles** seront représentés par le caractère « + ».

La liste **niveau_1** contiendra le labyrinthe.



Exercice 11 : Labyrinthe

1. **Créer** un nouveau script et **enregistrer**-le sous le nom « **lab_1.py** ».
2. **Editer** le script qui doit définir la liste « **niveau_1** » dont les éléments doivent permettre de représenter le labyrinthe ci-dessus et afficher le labyrinthe.