Formatage de chaîne en Python

juin 16, 2020 Aucun commentaire fonction format(), format, python

Python utilise le style du langage C pour créer de nouvelles chaînes formatées.

L'opérateur « % » est utilisé pour formater un ensemble de variables contenues dans un « tuple, ainsi qu'une chaîne qui contient du texte normal avec des symboles spéciaux comme « %s » et « %d ».
Disons que vous avez une variable appelée « name » et que vous souhaitez ensuite afficher un message.

```
name = "Thomas"
print("Hello %s, Welcome to WayToLearnX!" % name)
```

Sortie:

```
Hello Thomas, Welcome to WayToLearnX!
```

Si vous avez deux ou plusieurs arguments, utilisez un tuple (parenthèses):

```
name = "Thomas"
age = 18
print("My name is %s, I am %d years old." % (name, age))
```

Sortie:

```
My name is Thomas, I am 18 years old.
```

Tout objet qui n'est pas une chaîne peut également être formaté à l'aide de l'opérateur %s. Par exemple:

```
dict = {1:"A", 2:"B", 3:"C"}
print("My dictionnary: %s" % dict)
```

Sortie:

```
My dictionnary: {1: 'A', 2: 'B', 3: 'C'}
```

Voici quelques opérateurs de base que vous devez connaître:

- %s – Chaîne (ou tout objet avec une représentation sous forme de chaîne)

- %d – Entiers
- %f – Nombres à virgule flottante

Vous pouvez aussi formater une chaîne en utilisant la méthode **format()**.

La méthode format()

La méthode **format()** vous permet de formater des parties sélectionnées d'une chaîne.

Parfois, il y a des parties d'un texte que vous ne contrôlez pas, peut-être, elles proviennent d'une base de données ou une entrée d'un utilisateur.

Pour contrôler ces valeurs, ajoutez des accolades {} dans le texte et exécutez les valeurs via la méthode **format()**:

Exemple:

```
name = "Thomas"
str = "Hello {}, Welcome to WayToLearnX!"
print(str.format(name))
```

Sortie:

```
Hello Thomas, Welcome to WayToLearnX!
```

Si vous souhaitez utiliser plus de valeurs, ajoutez simplement plus de valeurs à la méthode **format()**:

Exemple:

```
name = "Thomas"
age = 18
str = "My name is {}, I am {} years old."
print(str.format(name, age))
```

Sortie:

```
My name is Thomas, I am 18 years old.
```

Numéros d'index

Vous pouvez utiliser des numéros d'index (un nombre entre accolades {0}) pour vous assurer que les valeurs sont placées dans les espaces appropriés.

Exemple:

```
num = 4
rue = 63
quartier = 90
adresse = "N°{0}, Rue {1} Quartier {2} Paris."
print(adresse.format(num, rue, quartier))
```

Sortie:

```
N°4, Rue 63 Quartier 90 Paris.
```

Ensuite, si vous souhaitez faire référence à la même valeur plusieurs fois, utilisez le même numéro d'index.

Exemple:

```
heure = 9
adresse = "Du {0}h du matin, à {0}h du soir."
print(adresse.format(heure))
```

Sortie:

```
Du 9h du matin, à 9h du soir.
```

Date et Heure en Python

juin 15, 2020 Aucun commentaire

Python possède un module nommé **datetime** pour travailler avec les dates et les heures. Regardons quelques exemples.

Exemple 1: Récupérer la date et l'heure actuelles

```
import datetime

date = datetime.datetime.now()
```

```
print(date)
```

Lorsque vous exécutez le programme, la sortie sera quelque chose comme:

```
2021-02-15 11:44:57.395327
```

Ici, nous avons importé le module datetime à l'aide de l'instruction **import datetime**.
L'une des classes définies dans le module datetime est la classe datetime. Nous avons ensuite utilisé la méthode **now()** pour créer un objet datetime contenant la date et l'heure locales actuelles.

La date contient l'année, le mois, le jour, l'heure, la minute, la seconde et la microseconde.

Le module 'datetime' possède de nombreuses méthodes pour renvoyer des informations sur l'objet date.

L'exemple suivant renvoie l'année et le nom du jour de la semaine:

```
import datetime

date = datetime.datetime.now()

print(date.year)

print(date.strftime("%A"))
```

Lorsque vous exécutez le programme, la sortie sera quelque chose comme:

```
2021

Monday
```

Exemple 2: Récupérer la date actuelle

```
import datetime

date = datetime.date.today()

print(date)
```

Lorsque vous exécutez le programme, la sortie sera quelque chose comme:

```
2021-02-15
```

Dans ce programme, nous avons utilisé la méthode **today()** définie dans la classe 'date' pour obtenir un objet date contenant la date locale actuelle.

Créer un objet Date

Pour créer une date, nous pouvons utiliser le constructeur datetime() de la classe datetime du module datetime.

La classe datetime() nécessite trois paramètres pour créer une date: année, mois, jour.

L'exemple suivant crée un objet Date:

```
import datetime

date = datetime.datetime(2021, 6, 21)

print(date)
```

Sortie:

```
2021-06-21 00:00:00
```

Récupérer la date à partir d'un timestamp

Nous pouvons également créer des objets date à partir d'un timestamp. Un timestamp Unix est le nombre de secondes entre une date particulière et le 1er janvier 1970 à UTC. Vous pouvez convertir un timestamp en date à l'aide de la méthode **fromtimestamp()**.

```
from datetime import date

timestamp = date.fromtimestamp(1623759849)

print("Date =", timestamp)
```

Sortie:

```
Date = 2021-06-15
```

Créer un objet Time

Un objet time instancié à partir de la classe Time représente l'heure locale.

```
from datetime import time

t1 = time()

print("t1 = ", t1)

# time(hour, minute and second)

t2 = time(12, 50, 59)

print("t2 = ", t2)

# time(hour, minute and second)

t3 = time(hour = 12, minute = 50, second = 59)

print("t3 = ", t3)

# time(hour, minute, second, microsecond)

t4 = time(12, 50, 59, 263998)

print("t4 = ", t4)
```

Sortie:

```
t1 =  00:00:00

t2 =  12:50:59

t3 =  12:50:59

t4 =  12:50:59.263998
```

Afficher heure, minute, seconde et microseconde

Une fois que vous avez créé un objet time, vous pouvez facilement afficher ses attributs tels que l'heure, minute, seconde, etc.

```
from datetime import time

t = time(12, 50, 59)

print("heure = ", t.hour)

print("minute = ", t.minute)

print("seconde = ", t.second)

print("microseconde = ", t.microsecond)
```

Sortie:

```
heure =  12

minute =  50

seconde  =  59

microseconde =  0
```

Afficher l'année, le mois, l'heure, la minute, et la seconde

```
from datetime import datetime

t = datetime(2021, 12, 30, 12, 55, 59)

print("année =", t.year)

print("mois =", t.month)

print("heure =", t.hour)

print("minute =", t.minute)

print("seconde =", t.second)
```

Sortie:

```
année = 2021

mois = 12
```

```
heure = 12

minute = 55

seconde = 59
```

Les modules en Python

juin 15, 2020 Aucun commentaire connaitre les fonctions d un module python, creer un module python, from ... import

Un module vous permet d'organiser logiquement votre code Python. Le regroupement du code associé dans un module facilite la compréhension et l'utilisation du code. Un module est un objet Python avec des attributs que vous pouvez lier et référencer.
Simplement, un module est un fichier composé de code Python. Un module peut définir des fonctions, des classes et des variables. Un module peut également inclure du code exécutable.

Créer un module:

Pour créer un module, enregistrez simplement le code que vous voulez dans un fichier avec l'extension **.py**.

Exemple:

Enregistrez ce code dans un fichier nommé « my_module.py ». Vous pouvez nommer le fichier de module comme vous le souhaitez, mais il doit avoir l'extension de fichier .py

```
def sayWelcome(name):

print(name + ", Welcome to WayToLearnX!")
```

Utiliser un module

Maintenant, nous pouvons utiliser le module que nous venons de créer, en utilisant l'instruction **import**. L'exemple suivant importe le module nommé my_module et appele la fonction sayWelcome():

```
import my_module

my_module.sayWelcome("Thomas")
```

Sortie:

```
Thomas, Welcome to WayToLearnX!
```

Variables dans un module

Un module peut contenir des fonctions, mais aussi des variables de tous types (listes, tuples, dictionnaires, objets, etc...).

Exemple:

Enregistrez ce code dans un fichier nommé my_module.py

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]
```

Importez le module nommé my_module et accédez à la liste:

```
import my_module

color = my_module.liste[1]

print(color)
```

Sortie:

```
Red
```

Renommer un module

Vous pouvez créer un alias lorsque vous importez un module, en utilisant le mot clé **as**:

Exemple:

```
import my_module as msg
```

```
msg.sayWelcome("Thomas")
```

Sortie:

```
Thomas, Welcome to WayToLearnX!
```

Importer à partir du module

Vous pouvez choisir d'importer uniquement les parties d'un module, en utilisant le mot-clé **from**.

Exemple:

Le module nommé my_module a une fonction et une liste:

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]

def sayWelcome(name):

print(name + ", Welcome to WayToLearnX!")
```

L'exemple suivant importe uniquement la fonction sayWelcome() du module:

```
from my_module import sayWelcome

sayWelcome("Thomas")
```

Sortie:

```
Thomas, Welcome to WayToLearnX!
```

Remarque: lors de l'importation à l'aide du mot clé 'from', n'utilisez pas le nom du module lorsque vous faites référence à des éléments du module.

Modules intégrés

Il existe plusieurs modules intégrés en Python, que vous pouvez importer à tout moment.

Exemple:

Importez et utilisez le module 'math':

```
import math

x = math.sqrt(4)

print(x)
```

Sortie:

```
2.0
```

Utilisation de la fonction dir()

Il existe une fonction intégrée pour connaitre les fonctions (ou les variables) dans un module Python. La fonction **dir()**:

Exemple:

Lister tous les fonctions/variables définis dans le module 'math':

```
import math

x = dir(math)

print x
```

Sortie:

```
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',

'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',

'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',

'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',

'sqrt', 'tan', 'tanh']
```

Remarque: La fonction dir() peut être utilisée sur tous les modules, même ceux que vous créez vous-même.

Les tableaux en Python

juin 13, 2020 4 Commentaires les tableaux en python, remplir un tableau python, supprimer élément tableau, tableau python

En programmation, un tableau est une collection d'éléments du même type.

Les tableaux sont populaires dans la plupart des langages de programmation tels que Java, C/C++, JavaScript, etc. Cependant, en Python, ils ne sont pas si courants. Lorsque les gens parlent de tableaux en Python, le plus souvent, ils parlent de listes en Python. Si vous ne savez pas ce que sont les listes, vous devez absolument voir le tutoriel sur Liste en Python.

Les tableaux sont pris en charge en Python grâce au module « array ».

Différence entre Liste et Tableau en Python

Nous pouvons traiter les listes comme des tableaux. Cependant, le type d'éléments stockés est complètement différent. Par exemple:

```
#créer une liste avec des éléments de différents types

liste = ["A", 5, 2.2]
```

Si vous créez des tableaux à l'aide du module « array », tous les éléments du tableau doivent être du même type.

```
import array as arr

tableau = arr.array('d', ["A", 5, 2.2])
```

Sortie:

```
Traceback (most recent call last):

  File "main.py", line 6, in <module>

    tableau = arr.array('d', ["A", 5, 2.2])

TypeError: a float is required
```

Le code ci-dessus affiche une erreur, car la méthode **array()** attend un tableau de type float.

Comment créer un tableau en Python

Comme vous l'avez peut-être deviné à partir de l'exemple ci-dessus, nous devons importer le module « array » pour créer des tableaux. Par exemple:

```
import array as arr

tableau = arr.array('d', [1.0, 1.1, 1.2, 1.3])

print(tableau)
```

Sortie:

```
array('d', [1.0, 1.1, 1.2, 1.3])
```

Ici, nous avons créé un tableau de type float. La lettre 'd' est un code de type. Cela détermine le type du tableau lors de la création.

Les codes de type couramment utilisés sont listés comme suit:

Code	Type
b	signed char
B	unsigned char
h	signed short
H	unsigned short
l	signed long
L	unsigned long
i	int
f	float
d	double

Comment accéder aux éléments d'un tableau?

Vous accédez aux éléments du tableau en vous référant au numéro d'index.
L'exemple suivant affiche le deuxième élément du tableau:

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

print(tab[1])
```

Sortie:

```
2
```

Indexation négative

L'indexation négative signifie à partir de la fin, -1 se réfère au dernier élément, -2 se réfère à l'avant-dernier élément, etc. L'exemple suivant affiche le dernier élément du tableau:

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

print(tab[-1])
```

Sortie:

```
6
```

Plage d'index

Vous pouvez spécifier une plage d'index en spécifiant par où commencer et où terminer la plage.

Lors de la spécification d'une plage, la valeur de retour sera un nouveau tableau avec les éléments spécifiés.

L'exemple suivant renvoie le troisième et quatrième éléments:

```
import array as arr
```

```
tab = arr.array('i', [1, 2, 3, 4, 5, 6])

print(tab[2:4])
```

Sortie:

```
array('i', [3, 4])
```

La recherche commencera à l'index 2 (inclus) et se terminera à l'index 4 (non inclus).

En ignorant la valeur de départ, la plage commencera au premier élément. L'exemple suivant renvoie les éléments du début jusqu'à le troisième élément:

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

print(tab[:3])
```

Sortie:

```
array('i', [1, 2, 3])
```

En ignorant la valeur de fin, la plage ira à la fin du tableau. L'exemple suivant renvoie les éléments de l'index 2(troisième élément) jusqu'à la fin:

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

print(tab[2:])
```

Sortie:

```
array('i', [3, 4, 5, 6])
```

Plage d'index négative

Spécifiez des index négatifs si vous souhaitez commencer la recherche à la fin du tableau. L'exemple suivant renvoie les éléments de l'index -3 (inclus) à l'index -1 (exclus):

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

print(tab[-3:-1])
```

Sortie:

```
array('i', [4, 5])
```

Modifier la valeur d'un élément

Pour modifier la valeur d'un élément spécifique, referez-vous au numéro d'index. L'exemple suivant change le deuxième élément:

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

tab[1] = 100

print(tab)
```

Sortie:

```
array('i', [1, 100, 3, 4, 5, 6])
```

Parcourir un tableau en Python

Vous pouvez parcourir les éléments du tableau en utilisant la boucle for.
L'exemple suivant affiche tous les éléments du tableau, un par un:

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

for i in tab:

print(i)
```

Sortie:

```
1

2

3

4

5

6
```

Vous en apprendrez plus sur les boucles for dans notre chapitre Boucle for en Python.

Vérifiez si un élément existe dans un tableau

Pour déterminer si un élément spécifié est présent dans un tableau, utilisez le mot clé **in**. L'exemple suivant vérifie si le nombre 5 est présent dans le tableau:

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

if 5 in tab:

print("5 existe dans le tableau")
```

Sortie:

```
5 existe dans le tableau
```

Longueur d'un tableau

Pour déterminer le nombre d'éléments d'un tableau, utilisez la fonction **len()**. L'exemple suivant affiche le nombre d'éléments dans le tableau:

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

print(len(tab))
```

Sortie:

```
6
```

Ajouter des éléments au tableau

Pour ajouter un élément à la fin du tableau, utilisez la méthode **append()**.
L'exemple suivant ajoute le nombre 7 en utilisant la méthode **append()**:

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

tab.append(7)

print(tab)
```

Sortie:

```
array('i', [1, 2, 3, 4, 5, 6, 7])
```

Supprimer un élément du tableau

Il existe plusieurs méthodes pour supprimer des éléments d'un tableau:

1- remove():
La méthode **remove()** supprime l'élément spécifié:

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

tab.remove(4)

print(tab)
```

Sortie:

```
array('i', [1, 2, 3, 5, 6])
```

2- pop():
La méthode **pop()** supprime l'index spécifié, (ou le dernier élément si l'index n'est pas spécifié):

```
import array as arr

tab = arr.array('i', [1, 2, 3, 4, 5, 6])

tab.pop()

print(tab)
```

Sortie:

```
array('i', [1, 2, 3, 4, 5])
```

If...Else en Python

juin 13, 2020 condition, if python, python if else sur une ligne, python if ifelse else

La prise de décision est requise lorsque nous voulons exécuter un code uniquement si une certaine condition est remplie.
L'instruction **if...elif...else** est utilisée en Python pour la prise de décision.

Syntaxe de l'instruction if en Python

```
if condition:

    instruction(s)
```

Ici, le programme évalue la condition et exécutera les instructions uniquement si la condition est True.

Si la condition est False, les instructions ne sont pas exécutées.

En Python, le corps de l'instruction if est indiqué par l'indentation(espace au début d'une ligne). Le corps commence par une indentation et la première ligne non indentée marque la fin.

Python interprète les valeurs non nulles comme True. None et 0 sont interprétés comme False.

Python prend en charge les conditions logiques mathématiques:

```
Égale à:              a == b

N'est pas égal à:     a! = b

Inférieur à:          a < b

Inférieur ou égal à: a <= b

Supérieur à:          a > b

Supérieur ou égal à: a >= b
```

Exemple:

```
a = 5

b = 10

if b > a:

print("B est supérieur à A")
```

Sortie:

```
B est supérieur à A
```

Dans l'exemple ci-dessus, nous utilisons deux variables, a et b, qui sont utilisées dans l'instruction if pour tester si b est supérieur à a. Comme a est 5 et b est 10, on sait que 10 est supérieur à 5, et donc nous affichons sur l'écran que « B est supérieur à A ».

Le mot-clé Elif

Le mot-clé elif est une façon de dire **« Si les conditions précédentes n'étaient pas True, alors essayez cette condition »**.

Exemple:

```
a = 10

b = 5

if b > a:

print("B est supérieur à A")

elif b < a:

print("B est inférieur à A")
```

Sortie:

```
B est inférieur à A
```

Dans l'exemple ci-dessus, b est inférieur à a, donc la première condition n'est pas True, mais la condition elif est True, nous affichons donc sur l'écran que « B est inférieur à A ».

Le mot-clé Else

Le mot-clé else intercepte tout ce qui n'est pas intercepté par les conditions précédentes.

Exemple:

```
a = 5

b = 5

if b > a:

print("B est supérieur à A")

elif b < a:

print("B est inférieur à A")

else:

print("B est égale à A")
```

Sortie:

```
B est égale à A
```

Dans l'exemple ci-dessus, b est égale à a, donc la première condition n'est pas True, ainsi la condition elif n'est pas True, donc nous allons à la condition else et affichons sur l'écran que « B est égale à A ».

Vous pouvez également avoir Else sans Elif:

```
a = 5

b = 5

if b > a:

print("B est supérieur à A")

else:

print("B est égale à A")
```

Sortie:

```
B est égale à A
```

If en une seule ligne

Si vous n'avez qu'une seule instruction à exécuter, vous pouvez la mettre sur la même ligne que l'instruction if.

```
if a > b: print("A est supérieur à B")
```

If...Else en une seule ligne

Si vous n'avez qu'une seule instruction à exécuter, une pour if et une pour else, vous pouvez tout mettre sur la même ligne:

```
print("A est supérieur à B") if a > b else print("A est inférieur ou égale à B")
```

Cette technique est connue sous le nom d'**opérateurs ternaires**.

Vous pouvez également avoir plusieurs instructions else sur la même ligne:

```
print(">") if a > b else print("<") if a < b else print("=")
```

If imbriqué

Vous pouvez avoir des instructions if à l'intérieur des instructions if, cela s'appelle des instructions if imbriquées.

Exemple:

```
n = 10

if n >= 0:

if n == 0:

print("Zero")

else:

print("Nombre positif")

else:

print("Nombre négatif")
```

Sortie:

```
Nombre positif
```

Les Sets en Python

juin 9, 2020 Aucun commentaire ensemble python, set python francais, set()

Set(ou ensemble en français) est une collection d'éléments non ordonnée.

Chaque élément du Set est unique (pas de doublons) et doit être immuable (ne peut pas être modifié). Pourtant, un Set lui-même est modifiable. Nous pouvons y ajouter ou en supprimer des éléments.
Les Sets peuvent également être utilisés pour effectuer des opérations mathématiques comme l'union, l'intersection, la différence symétrique, etc.

En Python, les Sets sont écrits avec des accolades {}. L'exemple suivant crée un Set:

```
mySet = {"A", "B", "C"}

print(mySet)
```

Sortie:

```
{'A', 'C', 'B'}
```

Modifier la valeur d'un élément

Une fois un Set créé, vous ne pouvez pas modifier ses éléments, mais vous pouvez ajouter de nouveaux éléments.

Ajouter des éléments

Pour ajouter un élément à un Set, utilisez la méthode **add()**. Pour ajouter plusieurs éléments à un Set, utilisez la méthode **update()**.

L'exemple suivant ajoute un élément à un Set, en utilisant la méthode **add()**:

```
mySet = {"A", "B", "C"}

mySet.add("D")

print(mySet)
```

Sortie:

```
{'A', 'C', 'B', 'D'}
```

L'exemple suivant ajoute plusieurs éléments à un Set, en utilisant la méthode **update()**:

```
mySet = {"A", "B", "C"}

mySet.update(["D", "E", "F"])

print(mySet)
```

Sortie:

```
{'F', 'E', 'D', 'B', 'C', 'A'}
```

Vérifiez si un élément existe dans un Set

Pour déterminer si un élément spécifié est présent dans un Set, utilisez le mot clé **in**. L'exemple suivant vérifie si « B » est présent dans l'ensemble:

```
mySet = {"A", "B", "C"}

print("B" in mySet)
```

Sortie:

```
True
```

Parcourir un Set en Python

Vous pouvez parcourir les éléments d'un Set en utilisant la boucle for. L'exemple suivant affiche tous les éléments d'un Set, un par un:

```
mySet = {"A", "B", "C"}

for i in mySet:

print(i)
```

Sortie:

```
A

C

B
```

Vous en apprendrez plus sur les boucles for dans notre chapitre Boucle for en Python.

Longueur d'un Set

Pour déterminer le nombre d'éléments d'un Set, utilisez la fonction **len()**.
L'exemple suivant affiche le nombre d'éléments dans un Set:

```
mySet = {"A", "B", "C"}

print(len(mySet))
```

Sortie:

```
3
```

Supprimer un élément du Set

Pour supprimer un élément d'un Set, utilisez la méthode **remove()** ou **discard()**.

L'exemple suivant supprime l'élément « B » en utilisant la méthode **remove()**:

```
mySet = {"A", "B", "C"}

mySet.remove("B")

print(mySet)
```

Sortie:

```
{'A', 'C'}
```

L'exemple suivant supprime l'élément « B » en utilisant la méthode **discard()**:

```
mySet = {"A", "B", "C"}

mySet.discard("B")

print(mySet)
```

Sortie:

```
{'A', 'C'}
```

La différence entre la méthode **remove()** et **discard()** est la suivante: Si l'élément à supprimer n'existe pas, **remove()** déclenchera une erreur. Tandis que **discard()** ne déclenchera pas une erreur si l'élément à supprimer n'existe pas.
Vous pouvez également utiliser la méthode **pop()** pour supprimer un élément, mais cette méthode supprimera le dernier élément. N'oubliez pas que les Sets ne sont pas ordonnés, vous ne saurez donc pas quel élément sera supprimé.

La valeur de retour de la méthode **pop()** est l'élément supprimé.

```
mySet = {"A", "B", "C"}

x = mySet.pop()

print(x)

print(mySet)
```

Sortie:

```
C

{'B', 'A'}
```

La méthode **clear()** vide le Set:

```
mySet = {"A", "B", "C"}

mySet.clear()

print(mySet)
```

Sortie:

```
set()
```

Le mot clé **del** supprimera complètement le Set:

```
mySet = {"A", "B", "C"}

del mySet
```

Les dictionnaires en Python

juin 9, 2020 Aucun commentaire dictionnaire de dictionnaire python, dictionnaire imbrique python, dictionnaire python, récupérer la clé d un dictionnaire python

Un dictionnaire est une collection non ordonnée, modifiable et indexée. En Python, les dictionnaires sont écrits avec des accolades {}, et ils ont des clés et des valeurs.
L'exemple suivant crée et affiche un dictionnaire:

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
print(dictionnaire)
```

Sortie:

```
{1: 'Python', 2: 'PHP', 3: 'Java'}
```

Accès aux éléments

Vous pouvez accéder aux éléments d'un dictionnaire en vous référant à son clé, entre crochets []. L'exemple suivant récupère la valeur de la clé 1:

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
print(dictionnaire[1])
```

Sortie:

```
Python
```

Vous pouvez aussi utiliser la méthode **get()** qui vous donnera le même résultat:

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
print(dictionnaire.get(1))
```

Sortie:

```
Python
```

Changer les valeurs d'un dictionnaire

Vous pouvez modifier la valeur d'un élément spécifique en vous référant à son clé. L'exemple suivant change la valeur du clé 1:

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
dictionnaire[1] = "Django"
print(dictionnaire)
```

Sortie:

```
{1: 'Django', 2: 'PHP', 3: 'Java'}
```

Parcourir un dictionnaire en Python

Vous pouvez parcourir les éléments d'un dictionnaire en utilisant la boucle for. Lorsque vous parcourez un dictionnaire, les valeurs de retour sont les clés du dictionnaire, mais il existe également des méthodes pour renvoyer les valeurs. L'exemple suivant affiche tous les clés du dictionnaire :

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
for key in dictionnaire:
print(key)
```

Sortie:

```
1
2
3
```

L'exemple suivant affiche tous les valeurs du dictionnaire :

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
```

```
for key in dictionnaire:

print(dictionnaire[key])
```

Sortie:

```
Python

PHP

Java
```

Vous pouvez également utiliser la méthode **values()** pour renvoyer les valeurs d'un dictionnaire:

```
dictionnaire = {

1 : "Python",

2 : "PHP",

3 : "Java"

}

for val in dictionnaire.values():

print(val)
```

Sortie:

```
Python

PHP

Java
```

Vous pouvez parcourir les clés et les valeurs à l'aide de la méthode **items()**:

```
dictionnaire = {

1 : "Python",

2 : "PHP",

3 : "Java"
```

```
}
for key, value in dictionnaire.items():
print(key, value)
```

Sortie:

```
1 Python
2 PHP
3 Java
```

Vérifiez si une clé existe dans un dictionnaire

Pour déterminer si la clé spécifié est présent dans un dictionnaire, utilisez le mot clé **in**. L'exemple suivant vérifie si la clé 2 est présent dans le dictionnaire:

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
if 2 in dictionnaire:
print("2 existe dans le dictionnaire")
```

Sortie:

```
2 existe dans le dictionnaire
```

Longueur d'un dictionnaire

Pour déterminer le nombre d'éléments d'un dictionnaire, utilisez la fonction **len()**. L'exemple suivant affiche le nombre d'éléments dans le dictionnaire:

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
print(len(dictionnaire))
```

Sortie:

```
3
```

Ajouter des éléments au dictionnaire

L'ajout d'un élément au dictionnaire se fait en utilisant une nouvelle clé et en lui affectant une valeur:

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
dictionnaire[4] = "C++"
print(dictionnaire)
```

Sortie:

```
{1: 'Python', 2: 'PHP', 3: 'Java', 4: 'C++'}
```

Supprimer un élément du dictionnaire

Il existe plusieurs méthodes pour supprimer des éléments d'un dictionnaire:

1- pop()
La méthode **pop()** supprime l'élément avec la clé spécifié:

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
dictionnaire.pop(2)
print(dictionnaire)
```

Sortie:

```
{1: 'Python', 3: 'Java'}
```

2- popitem()
La méthode **popitem()** supprime le dernier élément inséré (dans les versions antérieures de Python 3.7, la méthode **popitem()** supprime un élément aléatoire):

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
dictionnaire.popitem()
print(dictionnaire)
```

Sortie:

```
{1: 'Python', 2: 'PHP'}
```

3- del

Le mot clé **del** supprime l'élément avec la clé spécifié:

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
del dictionnaire[2]
print(dictionnaire)
```

Sortie:

```
{1: 'Python', 3: 'Java'}
```

4- clear()

La méthode **clear()** vide le dictionnaire:

```
dictionnaire = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
dictionnaire.clear()
print(dictionnaire)
```

Sortie:

```
{ }
```

Dictionnaires imbriqués

Un dictionnaire peut également contenir des dictionnaires, c'est ce qu'on appelle des dictionnaires imbriqués.

```
persone = {
"p1" : {
"nom" : "Alex",
"age" : 18
},
"p2" : {
"nom" : "Thomas",
"age" : 25
},
"p3" : {
"nom" : "Yohan",
"age" : 44
}
}
```

Copier un dictionnaire

Vous ne pouvez pas copier un dictionnaire simplement en tapant dict2 = dict1, car dict2 ne sera qu'une référence à dict1, et les modifications apportées dans dict1 seront automatiquement apportées dans dict2.

Il existe un moyen de faire une copie, une façon consiste à utiliser la méthode de dictionnaire intégrée **copy()**.

```
dict1 = {
1 : "Python",
2 : "PHP",
3 : "Java"
}
dict2 = dict1.copy()
print(dict2)
```

Sortie:

```
{1: 'Python', 2: 'PHP', 3: 'Java'}
```

Les tuples en Python

juin 9, 2020 Aucun commentaire tuple python

Un tuple est une séquence immuable d'objets. Les tuples sont des séquences, tout comme les listes. La différence entre les tuples et les listes est, les tuples ne peuvent pas être modifiés contrairement aux listes et les tuples utilisent des parenthèses (), tandis que les listes utilisent des crochets [].
Créer un tuple est aussi simple, il suffit de mettre les valeurs séparées par des virgules. Vous pouvez également mettre ces valeurs séparées par des virgules entre parenthèses. Par exemple :

```
t1 = ('A', 'B', 'C', 1, 2, 3)
print(t1)
```

Sortie:

```
('A', 'B', 'C', 1, 2, 3)
```

```
t2 = 'A', 'B', 'C', 1, 2, 3

print(t2)
```

Sortie:

```
('A', 'B', 'C', 1, 2, 3)
```

Un tuple vide est écrit comme deux parenthèses ne contenant rien:

```
tuple = ()
```

Accéder aux éléments de tuple

Vous pouvez accéder aux éléments de tuple en vous référant au numéro d'index, entre crochets. L'exemple suivant affiche le deuxième élément du tuple:

```
tupl = ("PHP", "Pyton", "Java")

print(tupl[1])
```

Sortie:

```
Pyton
```

Indexation négative

L'indexation négative signifie à partir de la fin, -1 se réfère au dernier élément, -2 se réfère à l'avant-dernier élément, etc. L'exemple suivant affiche le dernier élément de tuple:

```
tupl = ("Blue", "Red", "Green")

print(tupl[-1])
```

Sortie:

```
Green
```

Plage d'index

Vous pouvez spécifier une plage d'index en spécifiant par où commencer et où terminer la plage.

Lors de la spécification d'une plage, la valeur de retour sera un nouvel tuple avec les éléments spécifiés.

L'exemple suivant renvoie le troisième et quatrième éléments:

```
tupl = ("Blue", "Red", "Green", "Orange", "Black", "Yellow")

print(tupl[2:4])
```

Sortie:

```
('Green', 'Orange')
```

La recherche commencera à l'index 2 (inclus) et se terminera à l'index 4 (non inclus).

En ignorant la valeur de départ, la plage commencera au premier élément. L'exemple suivant renvoie les éléments du début jusqu'à la chaîne « orange »:

```
tupl = ("Blue", "Red", "Green", "Orange", "Black", "Yellow")

print(tupl[:3])
```

Sortie:

```
('Blue', 'Red', 'Green')
```

En ignorant la valeur de fin, la plage ira à la fin du tuple. L'exemple suivant renvoie les éléments de « Green » jusqu'à la fin:

```
tupl = ("Blue", "Red", "Green", "Orange", "Black", "Yellow")

print(tupl[2:])
```

Sortie:

```
('Green', 'Orange', 'Black', 'Yellow')
```

Plage d'index négative

Spécifiez des index négatifs si vous souhaitez commencer la recherche à la fin du tuple. L'exemple suivant renvoie les éléments de l'index -3 (inclus) à l'index -1 (exclus):

```
tupl = ("Blue", "Red", "Green", "Orange", "Black", "Yellow")

print(tupl[-3:-1])
```

Sortie:

```
('Orange', 'Black')
```

Modifier la valeur d'un tuple en Python

Une fois un tuple créé, vous ne pouvez pas modifier ses valeurs. Les tuples sont immuables. Mais il y'a une astuce. Vous pouvez convertir le tuple en liste, modifier la liste et reconvertir la liste en tuple.

```
#créer un tuple

tupl = ("A", "B", "C")

#convertir tuple en liste

liste = list(tupl)

#modifier le 1er élément du liste

liste[0] = "X"

#convertir liste en tuple

tupl = tuple(liste)

#afficher le tuple

print(tupl)
```

Sortie:

```
('X', 'B', 'C')
```

Parcourir un tuple en Python

Vous pouvez parcourir les éléments du tuple en utilisant la boucle for. L'exemple suivant affiche tous les éléments du tuple, un par un:

```
tupl = ("Blue", "Red", "Green", "Orange", "Black", "Yellow")

for i in tupl:

print(i)
```

Sortie:

```
Blue

Red

Green

Orange

Black

Yellow
```

Vous en apprendrez plus sur les boucles for dans notre chapitre Boucle for en Python.

Vérifiez si un élément existe dans un tuple

Pour déterminer si un élément spécifié est présent dans un tuple, utilisez le mot clé **in**. L'exemple suivant vérifie si la couleur « Red » est présent dans le tuple:

```
tupl = ("Blue", "Red", "Green")

if "Red" in tupl:

print("'Red' existe dans le tuple")
```

Sortie:

```
'Red' existe dans le tuple
```

Longueur d'un tuple

Pour déterminer le nombre d'éléments d'un tuple, utilisez la fonction **len()**.
L'exemple suivant affiche le nombre d'éléments dans le tuple:

```
tupl = ("Blue", "Red", "Green")

print(len(tupl))
```

Sortie:

```
3
```

Ajouter un élément au tuple

Une fois un tuple créé, vous ne pouvez pas y ajouter d'éléments. Les tuples sont immuables.

Supprimer un élément du tuple

Les tuples sont immuables, vous ne pouvez donc pas en supprimer des éléments, mais vous pouvez supprimer complètement le tuple

L'exemple suivant supprime complètement le tuple en utilisant le mot clé **del**:

```
tupl = ("Blue", "Red", "Green")

del tupl
```

Les listes en Python

juin 8, 2020 Aucun commentaire afficher une liste, append, copier une liste, créer une liste vide python, del, liste de liste python, liste vide python, parcourir une liste python, remove, sous liste python, supprimer élément liste python

Python propose une gamme de types composés souvent appelés séquences.

Liste est l'un des types de données les plus fréquemment utilisés et les plus polyvalents utilisés en Python.

Comment créer une liste en Python?

En programmation Python, une liste est créée en plaçant tous les éléments entre crochets [], séparés par des virgules.

Il peut avoir n'importe quel nombre d'éléments et ils peuvent être de différents types (entier, flottant, chaîne, etc.).

```
# liste vide

liste = []

# liste d'entiers

liste = [1, 2, 3]

# liste avec des types de données mixtes

liste = [10, "WayToLearnX", 2.5]
```

Une liste peut également avoir une autre liste en tant qu'élément. Cela s'appelle une liste imbriquée.

```
# liste imbriquée

my_list = ["WayToLearnX", [1, 2, 3], 2.8, ['x']]
```

Comment accéder aux éléments d'une liste?

Vous accédez aux éléments de la liste en vous référant au numéro d'index. L'exemple suivant affiche le deuxième élément de la liste:

```
liste = ["Blue", "Red", "Green"]

print(liste[1])
```

Sortie:

```
Red
```

Indexation négative

L'indexation négative signifie à partir de la fin, -1 se réfère au dernier élément, -2 se réfère à l'avant-dernier élément, etc. L'exemple suivant affiche le dernier élément de la liste:

```
liste = ["Blue", "Red", "Green"]

print(liste[-1])
```

Sortie:

```
Green
```

Plage d'index

Vous pouvez spécifier une plage d'index en spécifiant par où commencer et où terminer la plage.

Lors de la spécification d'une plage, la valeur de retour sera une nouvelle liste avec les éléments spécifiés.

L'exemple suivant renvoie le troisième et quatrième éléments:

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]

print(liste[2:4])
```

Sortie:

```
['Green', 'Orange']
```

La recherche commencera à l'index 2 (inclus) et se terminera à l'index 4 (non inclus).

En ignorant la valeur de départ, la plage commencera au premier élément. L'exemple suivant renvoie les éléments du début jusqu'à la chaîne « orange »:

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]

print(liste[:3])
```

Sortie:

```
['Blue', 'Red', 'Green']
```

En ignorant la valeur de fin, la plage ira à la fin de la liste. L'exemple suivant renvoie les éléments de « Green » jusqu'à la fin:

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]

print(liste[2:])
```

Sortie:

```
['Green', 'Orange', 'Black', 'Yellow']
```

Plage d'index négative

Spécifiez des index négatifs si vous souhaitez commencer la recherche à la fin de la liste. L'exemple suivant renvoie les éléments de l'index -3 (inclus) à l'index -1 (exclus):

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]

print(liste[-3:-1])
```

Sortie:

```
['Orange', 'Black']
```

Modifier la valeur d'un élément

Pour modifier la valeur d'un élément spécifique, referez-vous au numéro d'index. L'exemple suivant change le deuxième élément:

```
liste = ["Blue", "Red", "Green"]
```

```
liste[1] = "Black"

print(liste)
```

Sortie:

```
['Blue', 'Black', 'Green']
```

Parcourir une liste en Python

Vous pouvez parcourir les éléments de la liste en utilisant la boucle for. L'exemple suivant affiche tous les éléments de la liste, un par un:

```
liste = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]

for i in liste:

print(i)
```

Sortie:

```
Blue

Red

Green

Orange

Black

Yellow
```

Vous en apprendrez plus sur les boucles for dans notre chapitre Boucle for en Python.

Vérifiez si un élément existe dans la liste

Pour déterminer si un élément spécifié est présent dans une liste, utilisez le mot clé **in**. L'exemple suivant vérifie si la couleur « Red » est présent dans la liste:

```
liste = ["Blue", "Red", "Green"]

if "Red" in liste:
```

```
print("'Red' existe dans la liste")
```

Sortie:

```
'Red' existe dans la liste
```

Longueur d'une liste

Pour déterminer le nombre d'éléments d'une liste, utilisez la fonction **len()**.
L'exemple suivant affiche le nombre d'éléments dans la liste:

```
liste = ["Blue", "Red", "Green"]

print(len(liste))
```

Sortie:

```
3
```

Ajouter des éléments à la liste

Pour ajouter un élément à la fin de la liste, utilisez la méthode **append()**.
L'exemple suivant ajoute l'élément « Black » en utilisant la méthode **append()**:

```
liste = ["Blue", "Red", "Green"]

liste.append("Black")

print(liste)
```

Sortie:

```
['Blue', 'Red', 'Green', 'Black']
```

Pour ajouter un élément à l'index spécifié, utilisez la méthode **insert()**. L'exemple suivant insère l'élément dans la troisième position:

```
liste = ["Blue", "Red", "Green"]
```

```
liste.insert(2, "Black")

print(liste)
```

Sortie:

```
['Blue', 'Red', 'Black', 'Green']
```

Supprimer un élément de la liste

Il existe plusieurs méthodes pour supprimer des éléments d'une liste:

1- remove():

La méthode **remove()** supprime l'élément spécifié:

```
liste = ["Blue", "Red", "Green"]

liste.remove("Red")

print(liste)
```

Sortie:

```
['Blue', 'Green']
```

2- pop():

La méthode **pop()** supprime l'index spécifié, (ou le dernier élément si l'index n'est pas spécifié):

```
liste = ["Blue", "Red", "Green"]

liste.pop()

print(liste)
```

Sortie:

```
['Blue', 'Red']
```

3- Le mot-clé del:
Le mot clé **del** supprime l'index spécifié:

```
liste = ["Blue", "Red", "Green"]

del liste[1]

print(liste)
```

Sortie:

```
['Blue', 'Green']
```

Le mot clé **del** peut également supprimer complètement la liste:

```
liste = ["Blue", "Red", "Green"]

del liste
```

4- clear():
La méthode **clear()** vide la liste:

```
liste = ["Blue", "Red", "Green"]

liste.clear()

print(liste)
```

Sortie:

```
[]
```

Concaténer deux listes

Il existe plusieurs façons de concaténer deux ou plusieurs listes en Python. L'un des moyens les plus simples consiste à utiliser l'opérateur +.

```
liste1 = ["A", "B" , "C"]

liste2 = [1, 2, 3]
```

```
liste3 = liste1 + liste2

print(liste3)
```

Sortie:

```
['A', 'B', 'C', 1, 2, 3]
```

Copier une liste

Vous ne pouvez pas copier une liste simplement en tapant liste2 = liste1, car liste2 ne sera qu'une référence à liste1, et les modifications apportées dans liste1 seront automatiquement apportées dans liste2.

Il existe un moyen de faire une copie, une façon consiste à utiliser la méthode de liste intégrée **copy()**.

```
liste1 = ["Blue", "Red", "Green"]

liste2 = liste1.copy()

print(liste2)
```

Sortie:

```
['Blue', 'Red', 'Green']
```

Les opérateurs en Python

juin 8, 2020 Aucun commentaire % en python, and, comment faire une addition sur python, comment faire une multiplication sur python, division, division entière, division float, exposant python, modulo python, not, operateur daffectation python, or, ou exclusif python, pourcentage en python, puissance, python opérateur logique, xor en python

Les opérateurs sont utilisés pour effectuer des opérations sur les variables et les valeurs.
La liste suivante décrive les différents opérateurs utilisés en Python.

- Opérateurs arithmétiques

- Opérateurs d'affectation
- Opérateurs de comparaison
- Opérateurs logiques
- Opérateurs d'identité
- Opérateurs d'appartenance
- Opérateurs binaires

Opérateurs arithmétiques Python

Les opérateurs arithmétiques Python sont utilisés avec des valeurs numériques pour effectuer des opérations arithmétiques courantes, telles que l'addition, la soustraction, la multiplication, la division, etc.

Opérateur	Nom	Exemple	Résultat
+	Addition	a + b	Somme de a et b
*	Multiplication	a * b	Produit de a et b
–	Soustraction	a – b	Différence de a et b
/	Division	a / b	Quotient de a et b
//	division entière	a // b	Résultat entier d'une division
%	Modulo	a % b	Reste de a divisé par b
**	Exposant	a ** b	a à la puissance b

Exemple:

```
a = 10
b = 5
print(a + b) # 15
print(a * b) # 50
print(a - b) # 5
print(a / b) # 2.0
print(a // b) # 2
print(a % b) # 0
print(a ** b) # 100000
```

Opérateurs d'affectation Python

Les opérateurs d'affectation Python sont utilisés avec des valeurs numériques pour écrire une valeur dans une variable.

Opérateur	Description	Exemple	Est la même que
=	Affectation simple	a = b	
+=	Addition puis affectation	a += b	a = a + b
-=	Soustraction puis affectation	a -= b	a = a – b
*=	Multiplication puis affectation	a *= b	a = a * b
/=	Division puis affectation	a /= b	a = a / b
%=	Modulo puis affectation	a %= $b	a = a % b
//=	Division entière puis affectation	a //= b	a = a // b
**=	Exponentiation puis affectation	a **= b	a = a ** b
&=	Et bit à bit puis affectation	a &= b	a = a & b
|=	Ou bit à bit puis affectation	a |= b	a = a | b
^=	XOR puis affectation	a ^= b	a = a ^ b
>>=	Décalage binaire à droite puis affectation	a >>= b	a = a >> b
<<=	Décalage binaire à gauche puis affectation	a <<= b	a = a << b

Exemple:

```
a = 10
a += 5
print(a) # 15
a -= 5
print(a) # 10
a *= 2
print(a) # 20
a /= 4
print(a) # 5.0
```

Opérateurs de comparaison Python

Les opérateurs de comparaison Python sont utilisés pour comparer deux valeurs (nombre ou chaîne de caractères) :

Opérateur	Nom	Exemple	Résultat
==	Égal à	a == b	Retourne True si a est égal à b
!=	Non égal à	a != b	Retourne True si a n'est pas égal à b
>	Supérieur à	a > b	Retourne True si a est supérieur à b
<	Inférieur à	a < b	Retourne True si a est inférieur à b
>=	Supérieur ou égal à	a >= b	Retourne True si a est supérieur ou égal à b

Opérateur	Nom	Exemple	Résultat
<=	Inférieur ou égal à	a <= b	Retourne True si a est inférieur ou égal à b

Exemple:

```
a = 1
b = 2
print(a == b) # False
print(a != b) # True
print(a > b) # False
print(a < b) # True
print(a >= b) # False
print(a <= b) # True
```

Opérateurs logiques Python

Les opérateurs logiques Python sont utilisés pour combiner des instructions conditionnelles.

Opérateur	Exemple	Résultat
and	a and b	True si les deux a et b sont Trues
or	a or b	True si a ou b est True
not	!a	True si a n'est pas True

Exemple:

```
a = 4
print(a > 3 and a < 5) # True
print(a > 3 or a < 5) # True
print(not(a > 3 and a < 5)) # False
```

Opérateurs d'identité Python

Les opérateurs d'identité sont utilisés pour comparer les objets, non pas s'ils sont égaux, mais s'ils sont en fait le même objet, avec le même emplacement mémoire:

Opérateur	Description	Exemple
is	Renvoie True si les deux variables sont le même objet	a is b
is not	Renvoie True si les deux variables ne sont pas le même objet	a is not b

Exemple:

```
a = ["blue", "red"]
b = ["blue", "red"]
c = a
# renvoie False car a n'est pas le même objet que b, même s'ils ont le même contenu
print(a is b)
# renvoie True car c est le même objet que a
print(a is c)
# renvoie True car a n'est pas le même objet que b, même s'ils ont le même contenu
print(a is not b)
```

Opérateurs d'appartenance Python

Les opérateurs d'appartenance sont utilisés pour tester si une séquence est présentée dans un objet:

Opérateur	Description	Exemple
in	Renvoie True si une séquence avec la valeur spécifiée est présente dans l'objet	a in b
not in	Renvoie True si une séquence avec la valeur spécifiée n'est pas présente dans l'objet	a not in b

Exemple:

```
a = ["blue", "red"]
print("red" in a) # True
print("red" not in a) # False
```

Opérateurs binaires en Python

Les opérateurs binaires en Python sont utilisés pour comparer les nombres (binaires):

Opérateur	Nom	Exemple	Résultat
&	AND	a & b	Met chaque bit à 1 si les deux bits sont à 1
|	OR	a | b	Définit chaque bit sur 1 si l'un des deux bits vaut 1
^	XOR	a ^ b	Définit chaque bit sur 1 si un seul des deux bits vaut 1
!=	Inégalité	a != b	Retourne True si a n'est pas égal à b
~	NOT	a ~ b	Inverse tous les bits
<<	Décalage vers la gauche	a << b	Décalage vers la gauche
>>	Décalage vers la droite	a >> b	Décalage vers la droite

Exercice Python Corrigé – Partie 1

avril 20, 2020 Aucun commentaire challenges de programmation, corrigé, défi programmation, entretien, exemple test, langage python test, python pratique, recrutement, solution, test en ligne, test technique python, tp python

Avec des exercices corrigés en Python, vous pratiquerez divers concepts du langage Python. Vous commencerez par des exercices Python de base à des exercices plus avancés. La solution est fournie pour chaque exercice. Vous devez essayer de résoudre chaque problème par vous-même avant de vérifier la solution. Si vous avez des questions concernant chaque problème, nous vous encourageons à les poster sur notre forum.

Exercice 1:

Créez une fonction qui prend deux nombres comme arguments et retourne leur somme.

Exemple:

```
somme(1, 2) ➞ 3


somme(4, 4) ➞ 8


somme(-1, -1) ➞ -2
```

Corrigé

Solution:

```
def somme(a, b):
return a + b
```

Exercice 2:

Créez une fonction qui prend deux nombres comme arguments et faire la soustraction.

Exemple:

```
soustraction(2, 1) → 1


soustraction(4, 2) → 2


soustraction(-3, -3) → 0
```

Corrigé
Solution:

```
def soustraction(a, b):
return a - b
```

Exercice 3:

Écrivez une fonction qui prend un nombre entier de minutes et le convertit en secondes.

Exemple:

```
convert(1) → 60


convert(2) → 120


convert(6) → 360
```

Corrigé
Solution:

```
def convert(minutes):
return minutes * 60
```

Exercice 4:

Écrivez une fonction qui convertit les heures en secondes.

Exemple:

```
convert(1) → 3600


convert(2) → 7200


convert(6) → 21600
```

Corrigé
Solution:

```
def convert(heures):
return heures * 3600
```

Exercice 5:

Créez une fonction qui prend la hauteur et la largeur et trouve le périmètre d'un rectangle.

Exemple:

```
getPerimeter(2, 4) → 12


getPerimeter(6, 10) → 32


getPerimeter(3, 6) → 18
```

Corrigé
Solution:

```
def getPerimeter(hauteur, largeur):
return (hauteur + largeur) * 2
```

Exercice 6: Le problème du fermier

Dans ce défi, un fermier vous demande de lui dire combien de pattes peuvent être comptées parmi tous ses animaux. Il y a trois espèces:

- poulets = 2 pattes
- vaches = 4 pattes
- chevaux = 4 pattes

Le fermier a compté ses animaux et il vous donne un sous-total pour chaque espèce. Vous devez implémenter une fonction qui renvoie le nombre total de pattes de tous les animaux.

L'ordre des animaux transmis à la fonction est **nbrsPattes(poulets, vaches, chevaux)**.

Exemple:
nbrsPattes(1, 4, 2) ⟶ 26

nbrsPattes(2, 2, 2) ⟶ 20

nbrsPattes(2, 0, 3) ⟶ 16

N'oubliez pas que le fermier veut connaître le nombre total de pattes et non pas le nombre total d'animaux.

Corrigé
Solution:

```
def nbrsPattes(poulets, vaches, chevaux):
return poulets * 2 + (vaches + chevaux) * 4;
```

Exercice 7:

Créez une fonction qui prend une liste de nombres, et renvoie le plus grand nombre de la liste.

Exemple:

```
max([6, 9, 1, 2]) ⟶ 9


max([10, 66, 12, 98]) ⟶ 98
```

```
max([1, 1, 1, 1, 1]) ➞ 1
```

Corrigé
Solution:

```
def getMax(n):
return max(n)
```

Exercice 8:

Créez une fonction qui prend une liste et renvoie la somme de tous les nombres de la liste.

Exemple:

```
sommeL([1, 2, 3, 4]) ➞ 10


sommeL([0, 0, 0, 1]) ➞ 1


sommeL([-1, -2, 3]) ➞ 0
```

Corrigé
Solution:

```
def sommeL(l):
return sum(l)
```

Exercice 9:

Créez une fonction qui prend une liste de nombres et renvoie le plus petit nombre de la liste.

Exemple:

```
getMin([9, 6, 1, 2]) ➞ 1


getMin([8, 8, 8, 8]) ➞ 8
```

```
getMin([-2, -8, -1]) → -8
```

Corrigé

Solution:

```
def getMin(l):
return min(l)
```

Exercice Python Corrigé – Partie 2

avril 20, 2020 Aucun commentaire challenges de programmation, corrigé, défi programmation, entretien, exemple test, langage python test, python pratique, recrutement, solution, test en ligne, test technique python, tp python

Avec des exercices corrigés en Python, vous pratiquerez divers concepts du langage Python. Vous commencerez par des exercices Python de base à des exercices plus avancés. La solution est fournie pour chaque exercice. Vous devez essayer de résoudre chaque problème par vous-même avant de vérifier la solution. Si vous avez des questions concernant chaque problème, nous vous encourageons à les poster sur notre forum.

Exercice 1:

Créez une fonction qui renvoie True si un entier est divisible par 5, sinon renvoie False.

Exemple:

```
isDivisibleBy5(5) → True

isDivisibleBy5(-5) → True

isDivisibleBy5(3) → False
```

Corrigé

Solution:

```
def isDivisibleBy5(n):
return not n % 5
```

Exercice 2:

Créez une fonction qui accepte une liste et renvoie le dernier élément de la liste.

Exemple:

```
getLast([1, 2, 3]) → 3


getLast(["A", "B", "C"]) → "C"


getLast([10, "WayToLearnX", True]) → True
```

Corrigé

Solution:

```
def getLast(liste):
return liste[-1]
```

Exercice 3:

Corrigez le code suivant (Erreur de syntaxe), pour calculer le carré.

Exemple:
carre(2) → 4

carre(4) → 16

Corrigé

Solution:

```
def carre(b):
return b ** 2
```

Exercice 4:

Créez une fonction qui évalue une équation.

Exemple:

```
evalute(1+2+3) → 6


evalute(4*2/2) → 4.0


evalute((3+2)*(3+2)) → 25
```

Corrigé

Solution:

```
def evalute(op):
return op
```

Exercice 5:

Créez une fonction qui renvoie la valeur ASCII du caractère transmis.

Exemple:

```
charToAscii("A") → 65


charToAscii("a") → 97


charToAscii("+") → 43
```

Corrigé

Solution:

```
def charToAscii(c):
return ord(c)
```

Exercice 6:

Créez une fonction qui renvoie True si une chaîne contient des espaces.

Exemple:

```
containSpaces("WayToLearnX") → False

containSpaces("Welcome to WayToLearnX") → True

containSpaces(" ") → True
```

Corrigé

Solution:

```
def containSpaces(str):
return " " in str
```

Exercice 7:

Créez une fonction qui prend un mot et détermine s'il est pluriel ou singulier. Un mot pluriel est celui qui se termine par « s ». S'il est pluriel renvoyer TRUE sinon FALSE.

Exemple:

```
checkIsPlural("enfants") → True

checkIsPlural("filles") → True
```

```
checkIsPlural("fille") → False


checkIsPlural("enfant") → False
```

Corrigé

Solution:

```
def checkIsPlural(str):
return str.endswith('s')
```

Exercice 8: Nombre paire ou impaire

Créez une fonction qui prend un nombre comme argument et renvoie « pair » pour les nombres pairs et « impair » pour les nombres impairs.

Exemple:

```
check(2) → "pair"


check(7) → "impair"


check(22) → "pair"
```

Corrigé

Solution:

```
def check(n):
return 'impair' if n % 2 else 'pair'
```

Exercice 9:

Créez une fonction qui renvoie True si une chaîne est vide, sinon False.

Exemple:

```
isEmpty("WayToLearnX") ➞ False


isEmpty(" ") ➞ False


isEmpty("") ➞ True
```

Corrigé

Solution:

```
def isEmpty(str):
return not str
```

Exercice Python Corrigé – Partie 3

avril 21, 2020 1 Commentaire challenges de programmation, corrigé, défi programmation, entretien, exemple test, langage python test, python pratique, recrutement, solution, test en ligne, test technique python, tp python

Avec des exercices corrigés en Python, vous pratiquerez divers concepts du langage Python. Vous commencerez par des exercices Python de base à des exercices plus avancés. La solution est fournie pour chaque exercice. Vous devez essayer de résoudre chaque problème par vous-même avant de vérifier la solution. Si vous avez des questions concernant chaque problème, nous vous encourageons à les poster sur notre forum.

Exercice 1:

Créez une fonction qui renvoie le nombre de valeurs « True » qu'il y a dans une liste.

Exemple:

```
count([False, False, True, True, True]) ➞ 3
```

```
count([False, False, False]) → 0


count([]) → 0
```

Corrigé

Solution:

```
def count(liste):
return sum(liste)
```

Exercice 2:

Vous avez embauché trois commerciales et vous les payez. Créez une fonction qui prend trois nombres (le salaire horaire de chaque commerciale) et renvoie la différence entre le commerciale le mieux payé et le moins payé.

Exemple:

```
getDiff(200, 10, 90) → 190
//200 - 10 = 190


getDiff(56, 29, 16) → 40


getDiff(2, 10, 5) → 8
```

Corrigé

Solution:

```
def getDiff(*args):
return max(args) - min(args)
```

Exercice 3: Pont cassé

Créez une fonction qui valide si un pont est sûr à marcher (c'est-à-dire, qu'il n'a pas d'espace entre #).

Exemple:

```
marcher('###### ########') → False


marcher('########') → True


marcher('#') → True
```

Corrigé

Solution:

```
def marcher(pont):
return ' ' not in pont
```

Exercice 4:

Créez une fonction qui prend une liste de nombres et renvoie le minimum et le maximum dans une liste [Min, Max].

Exemple:

```
getMinMax([8, 1, 9, 2, 6]) → [1, 9]


getMinMax([22, 2]) → [2, 22]


getMinMax([5]) → [5, 5]
```

Corrigé

Solution:

```
def getMinMax(liste):

return [min(liste), max(liste)]
```

Exercice 5:

Créez une fonction qui prend un nombre (de 1 à 10) et renvoie une chaîne de tirets correspondante.

Exemple:

```
convert(2) → "--"


convert(6) → "------"


convert(4) → "----"
```

Corrigé

Solution:

```
def convert(n):

return "-" * n
```

Exercice 6:

Créez une fonction qui prend une chaîne (un nom aléatoire). Si le dernier caractère du nom est un « s », retournez TRUE, sinon retournez FALSE.

Exemple:

```
checkS("Thomas") → True


checkS("Ali") → False
```

```
checkS("Alex") ➞ False


checkS("Alvis") ➞ True
```

Corrigé

Solution:

```
def checkS(str):
return str.endswith('s')
```

Exercice 7:

Créer une fonction qui divise a et b (a / b).

Exemple:

```
div(4,2) ➞ 2.0


div(10,2) ➞ 5.0


div(1,0) ➞ "Erreur"
```

Corrigé

Solution:

```
def div(a, b):
return a/b if b!=0 else "Erreur"
```

Exercice 8:

Écrivez un programme Python pour renvoyer le reste de deux nombres. Il existe un seul opérateur en Python, capable de fournir le reste d'une division. Deux

nombres sont transmis comme paramètres. Le premier paramètre divisé par le deuxième paramètre.

Exemple:

```
resteDiv(1, 3) → 1


resteDiv(2, 4) → 2


resteDiv(3, 3) → 0
```

Corrigé

Solution:

```
def resteDiv(a, b):
return a % b
```

Exercice 9:

Créez une fonction qui prend une liste d'éléments et renvoie le premier et le dernier éléments sous forme de nouvelle liste.

Exemple:

```
getFirstLast([1, 2, 3, 4, 5, 6, 7]) → [1, 7]


getFirstLast(["A", "B", "C", "D"]) → ["A", "D"]


getFirstLast(["A", 2, True, None]) → ["A", None]
```

Corrigé

Solution:

```
def getFirstLast(liste):

return [liste[0], liste[-1]]
```