Sprintf() Avec Arduino | Imprimer Plusieurs Variables Sur Le Moniteur Série

sprintf() – Moche à dire, mais incroyablement utile à maîtriser. Sprintf() signifie « format d'impression de chaîne (ted) ». C'est votre fonction incontournable si vous recherchez un moyen plus efficace de combiner du texte et des variables dans une chaîne pour la sortie vers le moniteur série. J'entrerai dans les détails sous peu, mais voici la principale raison pour laquelle vous devriez maîtriser sprintf() :

Un code plus simple, plus clair et moins encombré.

À quel point est-il plus maigre, demandez-vous ?

Qu'est-ce qui vous prendrait **neuf** lignes de code avec les commandes Serial.print standard que vous pourriez faire en seulement **trois** en utilisant sprintf(). Ajoutez cela à l'ensemble d'un projet, et c'est un gros problème. Si cela vous semble être une connaissance utile, vous êtes au bon endroit. Dans cette leçon, vous apprendrez tout ce qui précède et un peu plus.

Prêt? Allons-y.

Get FREE access to our Arduino Error AI Solver
Use Now

Tout D'abord, Revisitons La Manière Maladroite D'imprimer Une Chaîne

Voici donc une question Arduino 101 pour vous. Comment imprimeriez-vous une chaîne en utilisant du texte et des variables en utilisant le bon vieux Serial.print() ? Supposons que vous souhaitiez imprimer cette ligne de texte sur le moniteur série :

« Les 3 burritos sont à 147,7 degrés F »

Dans cet exemple, considérez à la fois le nombre de burritos et la valeur de la température comme variables. En utilisant Serial.print(), vous écrivez généralement cinq lignes de code pour imprimer cette seule ligne de texte.

```
En série. print ( "Le " ) ; < police >< /font >
En série. imprimer ( numBurritos ) ; < police >< /font >
En série. print ( " les burritos sont " ) ; < police >< /font >
En série. imprimer ( tempStr ) ; < police >< /font >
En série. println ( " degrés F" ) ;
```

Maintenant, pour être clair, il n'y a rien de mal en soi à utiliser Serial.print() pour créer une chaîne. Ça fera l'affaire. Mais le Carbunkle de la grand-tante Gertrude est-il maladroit. En fait, utiliser Serial.print() pour créer une chaîne devient encore plus maladroit à mesure que vous ajoutez de variables.

Et si vous vouliez imprimer une ligne avec 4 variables insérées dans une chaîne comme celle-ci :

"Les 3 burritos mesurent 147,7 degrés F, pèsent 14 onces et ont été terminés il y a 3 minutes."

Pour chaque variable que vous ajoutez à la sortie, vous ajoutez *deux impressions en série supplémentaires dans le code.* Donc, pour cette mise à jour utile et informative du burrito, vous regarderiez le canon de **neuf** lignes de code.

Sprintf() À La Rescousse

C'est là que sprintf() s'avère utile. Nous pouvons imprimer autant de variables que nous le souhaitons dans notre chaîne, et la quantité de code requise reste toujours à trois lignes gérables.

Voici les trois lignes de code dont vous aurez besoin :

```
 tampon de caractères [ 40 ] ; < police >< /font >
 sprintf ( buffer, "Les %d burritos font %s degrés F" , numBurritos, tempStr ) ; <
police >< /font >
 En série. println ( tampon ) ;
```

> *Membres de la Programming Electronics Academy, consultez le cours Arduino pour débutants absolus pour vous entraîner à utiliser la* ***bibliothèque série*** *dans votre code.*
>
> Pas encore membre? Inscrivez-vous ici.

Décomposons chaque ligne.

1. Tout d'abord, vous allez créer un tableau de caractères pour enregistrer la chaîne de sortie.
2. Ensuite, vous utilisez la fonction sprintf() pour combiner notre texte et nos variables dans une chaîne.
3. Enfin, vous demanderez à Serial.print() d'afficher la chaîne formatée.

Examinons de plus près chaque ligne de code.

```
tampon de caractères [ 40 ] ;
```

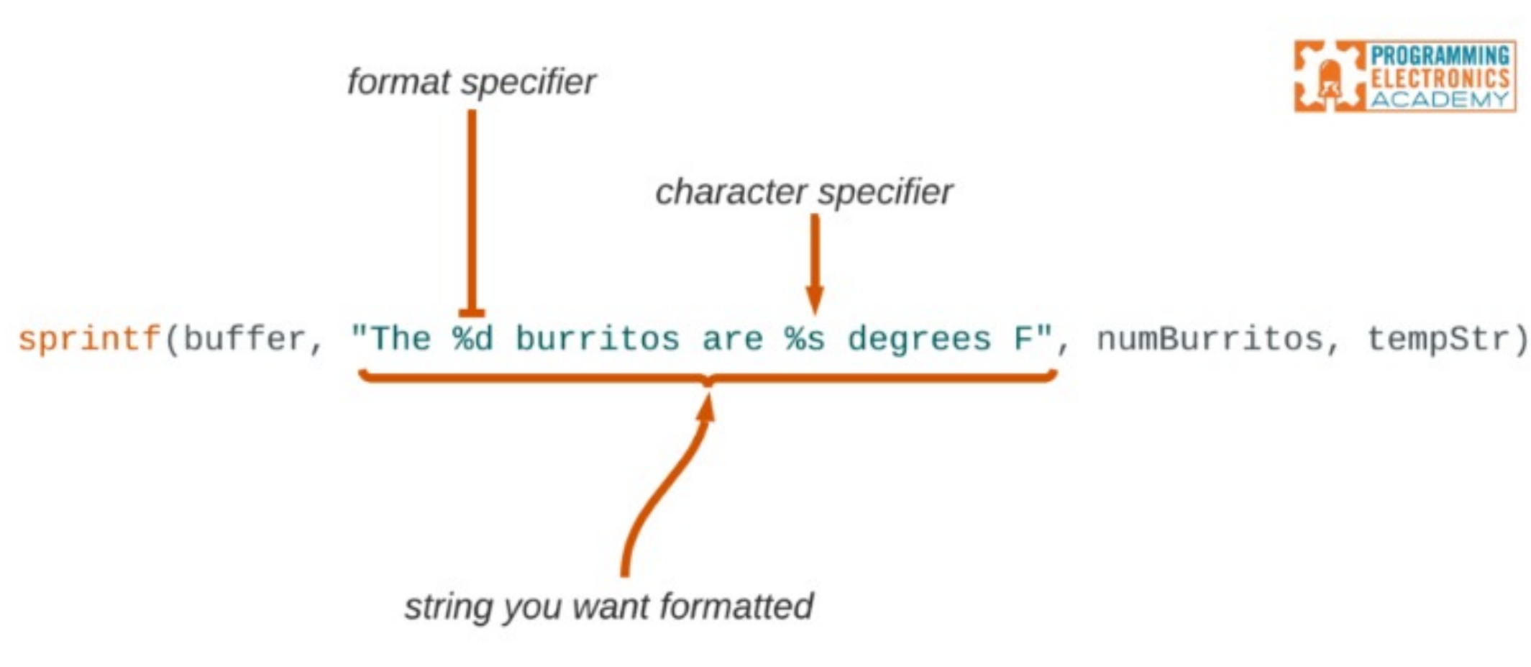
Le tableau de caractères doit être aussi grand ou plus grand que la chaîne de sortie finale. Comptez simplement les caractères que vous prévoyez de stocker dans cette chaîne et assurez-vous que le tampon est au moins aussi grand.

La ligne de code suivante est la fonction sprintf() réelle.

```
sprintf ( buffer, "Les %d burritos font %s degrés F" , numBurritos, tempStr ) ;
```

Notez que sprintf() nécessite un minimum de 2 arguments. Le premier argument est l'endroit où vous prévoyez de stocker la chaîne que sprintf() créera pour vous. C'est ici que vous utilisez le tampon de caractères que vous avez créé sur la ligne précédente.

L'argument suivant est la chaîne que vous souhaitez créer, remplie avec **les spécificateurs de format** dans lesquels vous souhaitez insérer vos variables. Le spécificateur de format est le signe %. La lettre qui suit le spécificateur de format est appelée **caractère de format** et indique à sprintf() quel type de données sera utilisé pour cette variable.



```
"Les 3 burritos font 147,7 degrés F"
```

Dans cet exemple, nous avons deux spécificateurs de format (%). Cela signifie que nous voulons que deux variables soient insérées dans la chaîne de sortie. Les spécificateurs de caractères sont un peu bizarres au début. Ce sont simplement des lettres qui représentent le type de données qui sera inséré.

Cela commence à avoir beaucoup plus de sens une fois que vous avez appris la signification de chaque lettre, alors regardons cela maintenant.

> *Membres de la Programming Electronics Academy, consultez le cours Arduino pour débutants absolus pour démarrer vos compétences en programmation Arduino.*
>
> Pas encore membre? Inscrivez-vous ici.

Spécificateurs De Caractères

Voici quelques-uns des spécificateurs de caractères courants :

d ou **i** – entier décimal signé
u – entier décimal non signé
s – une chaîne de caractères

Par exemple, si vous utilisez **%d** , vous dites à sprintf() de formater la variable insérée sous forme d'entier décimal signé.

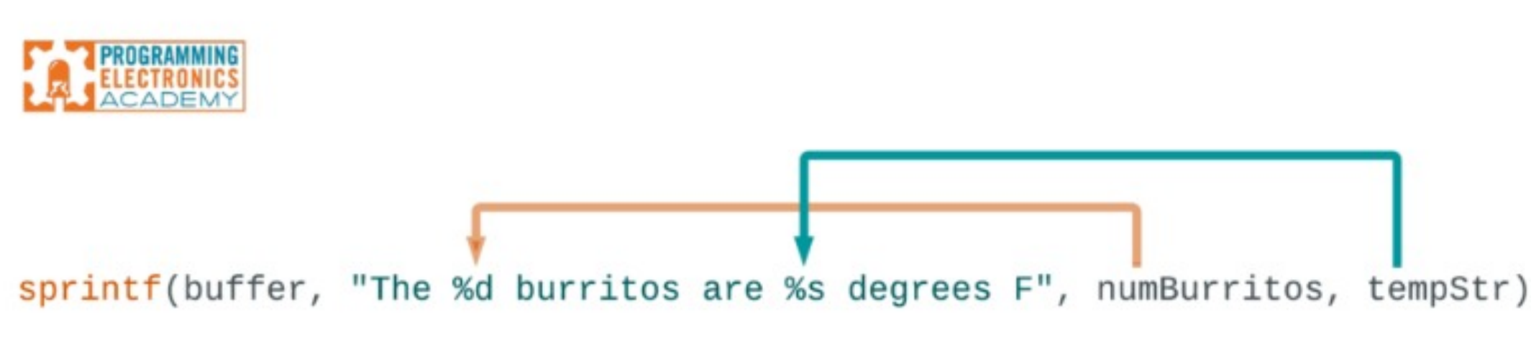
Vous vous demandez peut-être ce que cela signifie lorsqu'un caractère est défini comme un **entier décimal signé** . Voici le scoop :

- **Signé** signifie qu'il peut être positif ou négatif.
- **Décimal** signifie que nous voulons qu'il apparaisse sous forme décimale, au lieu d'être formaté en octal ou hexadécimal.
- **Un nombre entier** signifie qu'il s'agit d'un nombre entier, c'est-à-dire qu'il n'y a pas de virgule décimale.

Maintenant, où sprintf() trouve-t-il réellement les variables à insérer ? Eh bien, nous n'avons pas besoin de chercher bien loin, car ce sont les arguments ajoutés juste après la chaîne.

```
sprintf ( buffer, "Les %d burritos font %s degrés F" , numBurritos, tempStr ) ;
```

Pour chaque spécificateur de format, vous devez transmettre une valeur correspondante. Ces valeurs sont ajoutées comme arguments supplémentaires à sprintf(), chacune séparée par une virgule.



Dans cet exemple, nous avons deux spécificateurs de format, et donc deux arguments à la fin. Le premier, numBurritos, est inséré au niveau du premier spécificateur de format. Le second, tempStr, est inséré au niveau du deuxième spécificateur de format.

Si nous avions plus de spécificateurs de format dans notre chaîne, nous aurions besoin d'ajouter plus d'arguments à sprint(). Cependant, tout se déroule sur cette seule ligne de code. C'est ce qui fait de sprintf() une fonction si utile à ajouter à votre boîte à outils de codage.

Qu'en Est-Il Des Nombres À Virgule Flottante ?

Maintenant, vous pourriez vous dire… *« Attendez une seconde maintenant – je pensais que vous aviez dit que le formateur de caractères « s » était destiné à une chaîne de caractères, mais la température en degrés Fahrenheit est une valeur à virgule flottante – qu'est-ce que ça donne ?!*

Eh bien, voici l'affaire.

Sprintf() avec Arduino ne peut pas gérer les valeurs à virgule flottante. Donc, si vous devez imprimer quelque chose qui a un point décimal, comme 3,14 ou 156,7, vous devez d'abord convertir cette valeur flottante en chaîne de caractères, puis imprimer la chaîne. Un moyen pratique de le faire consiste à utiliser dtostrf() , qui convertit une valeur à virgule flottante en chaîne. Je n'entrerai pas dans les détails maintenant, mais n'oubliez pas de regarder notre autre vidéo sur l'utilisation de dtostrf() avec Arduino.

La dernière ligne de code de notre trio est le bon vieux Serial.print(). Ce que nous passons en argument est le tampon de caractères dans lequel sprintf() a stocké notre chaîne formatée. Vous remarquerez que sprintf() ne renvoie pas la chaîne elle-même. Il enregistre cette chaîne dans le tampon de caractères que nous avons spécifié. C'est pourquoi il suffit d'imprimer le contenu du buffer pour afficher notre chaîne.

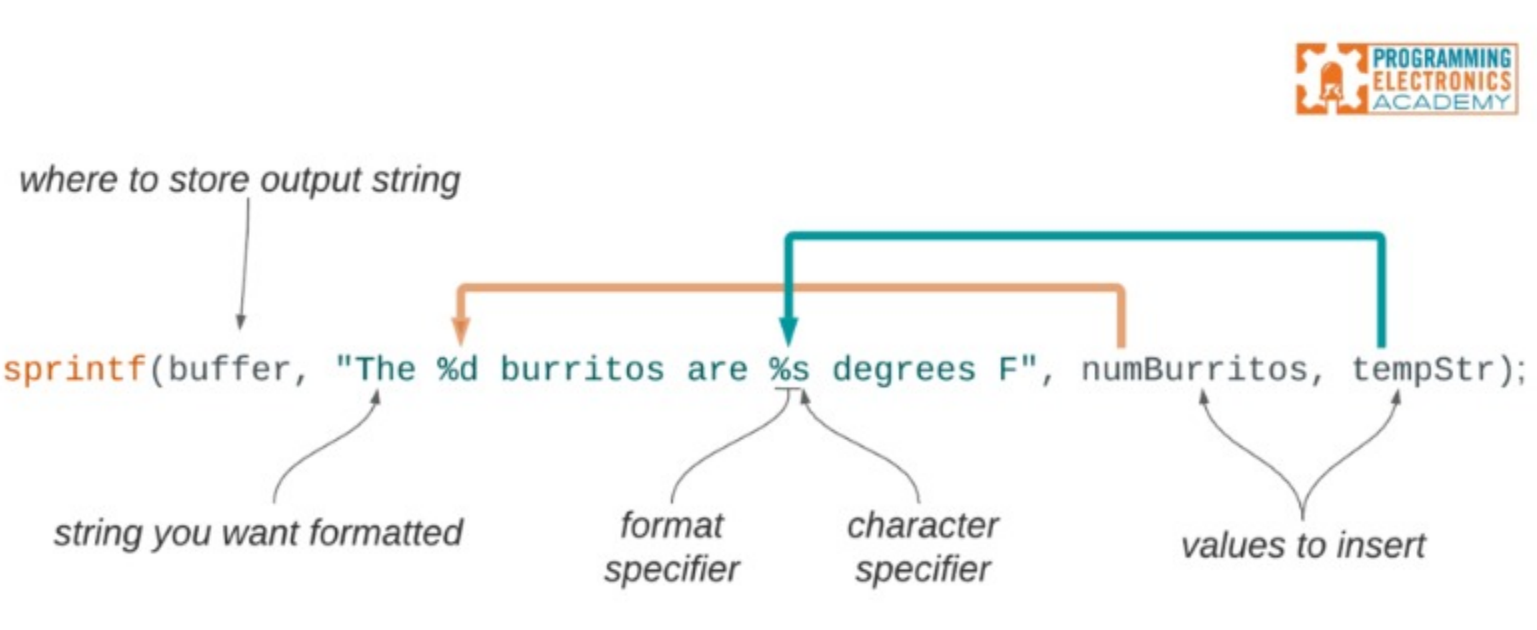
Ouf. OK, alors oui, nommons l'éléphant dans la pièce ; les nombres flottants sont un peu plus compliqués que de travailler avec d'autres valeurs. Cependant, l'incitation à utiliser l'approche sprintf() s'applique toujours : une fois que vous vous serez habitué à l'utiliser, vous écrirez un code plus simple et de meilleure qualité.

Oh, et avant de passer à une revue rapide de tout ce que vous avez appris, voici un détail intéressant à garder au fond de votre esprit. Il convient de noter que sprintf() renvoie une valeur si vous choisissez de l'utiliser, qui correspond au nombre total de caractères qui ont été stockés dans le tampon, à l'exclusion du caractère de fin nul ajouté par sprintf().

Une Revue Rapide De Sprintf()

OK, faisons une revue de 30 secondes de sprintf().

Voici une ligne de code utilisant la fonction sprint(f) :



Voici ce que cela fait :

1. La première valeur attendue par sprintf() est un tampon de caractères. C'est ici que la chaîne formatée sera stockée.
2. La deuxième valeur de sprintf() est la chaîne que vous souhaitez formater, y compris les éventuels spécificateurs de format.
3. Les arguments finaux sont les valeurs qui remplaceront les spécificateurs de format.

Et si vous vous sentez ambitieux…

Envie d'encore plus d'efficacité ? Vous êtes un animal de productivité ! (Je veux dire cela dans le bon sens.) Croyez-le ou non, il y a une tonne de choses supplémentaires que vous pouvez faire avec sprintf(). Essayez de jouer avec **les sous-spécificateurs.** Ces éléments facultatifs se trouvent entre le signe % et le spécificateur de caractère. Vous pouvez utiliser des sous-spécificateurs pour exécuter des fonctions de formatage utiles, telles que justifier à gauche les valeurs insérées et ajouter des zéros non significatifs.