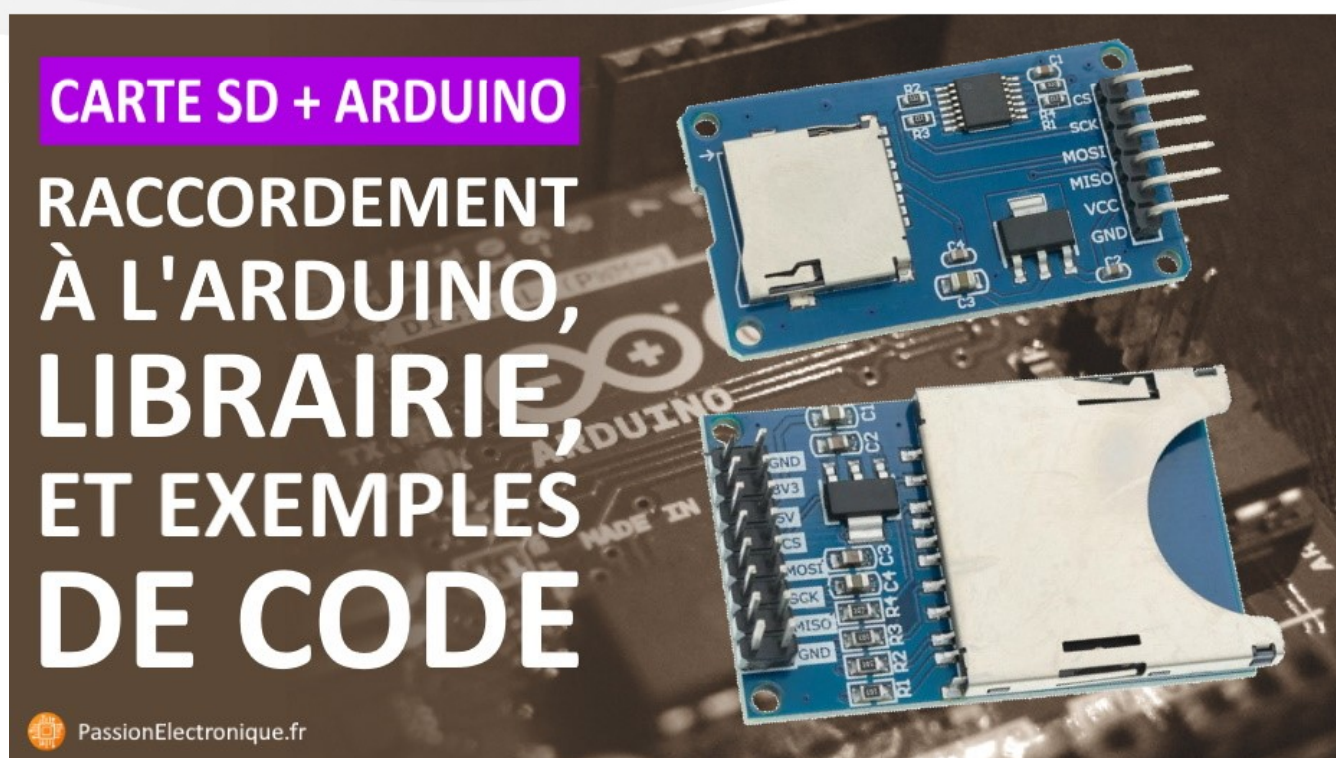




[Accueil](#) > [Arduino](#) > Carte SD Arduino : branchement, librairie de base, et exemples de code (pour Arduino Uno, Nano, Mega)

## Carte SD Arduino : branchement, librairie de base, et exemples de code (pour Arduino Uno, Nano, Mega)



Si vous cherchez un **moyen simple et rapide pour stocker un grand nombre d'informations depuis votre Arduino**, alors rien de tel qu'une **carte SD** pour ce faire ! Surtout qu'il existe maintenant des modules « prêts à l'emploi » et peu coûteux pour y arriver ! D'ailleurs, c'est ce que je vous propose de découvrir ici. Ainsi, vous verrez **comment raccorder un lecteur de carte micro SD à son Arduino** Uno, Nano, ou Mega, et **comment effectuer des lectures et écritures dans des fichiers sur une SD card**.

Histoire de vous montrer des applications concrètes de tout cela, je vous mettrai **plusieurs exemples de code pratiques, avec notamment celui d'un datalogger** (programme permettant l'enregistrement de données sur carte SD). Ce dernier vous permettra d'ailleurs de faire un suivi régulier de la température et de l'hygrométrie ambiante, via un capteur DHT22. Ces mesures seront stockées sur carte SD, et nous verrons comment elles peuvent être utilisées pour faire de beaux graphiques, dans un tableur (Excel, OpenOffice Calc, ...). Mais pour l'heure, commençons par les bases 😊

Ce contenu vous plaît ? Alors abonnez-vous à la Newsletter pour ne rien louper !

 Recevoir la Newsletter !

Si vous constatez la moindre coquille ou avez des questions, n'hésitez pas à m'en faire part en zone de commentaire, en bas de cet article ! Car cela me permettra d'améliorer la qualité du présent tuto, et d'apporter un maximum d'infos au plus grand nombre. Merci !



## Sommaire [\[ Masquer \]](#)

1. Survol rapide
2. Branchement et câblage d'une micro SD card à un Arduino Uno, Nano, ou Mega (port SPI)
3. La librairie SD.h (bibliothèque arduino)
4. Exemple de code #1 : récupérer les infos d'une carte SD (taille, format, ...)
5. Exemple de code #2 : lire et écrire un fichier dans une SD card (read/write/dump)
6. Exemple de code #3 : enregistrer des mesures de température et hygrométrie avec un DHT22 (data logger)
7. Exemple de code #4 : génération d'un nom de fichier aléatoire, et enregistrement sur carte SD
8. Carte SD Arduino : conclusion !

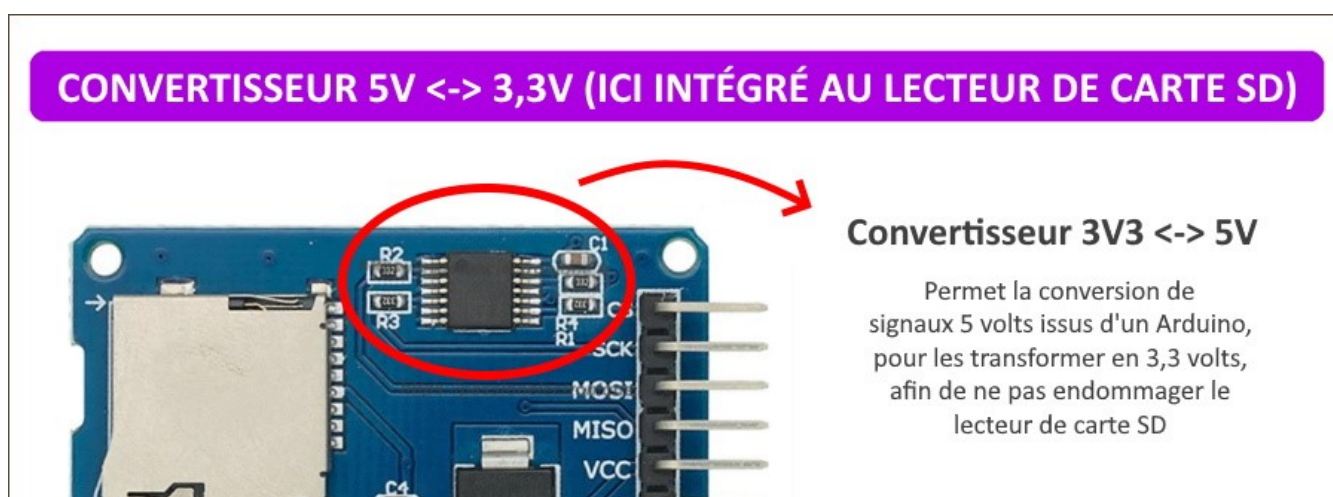
## 1. Survol rapide

Lorsque je parle de carte SD ici, je veux avant tout parler de **lecteur de carte SD raccordable à l'Arduino**. Prenant souvent la forme d'une mini plaquette PCB avec un support de SD card intégré, ces types de modules disposent la plupart du temps d'une interface de communication idéale pour communiquer avec un Arduino Uno, Nano, Mega, ou tout autre contrôleur disposant d'un port SPI. Physiquement, voici deux exemples de modules, qu'on trouve couramment en vente sur le marché :

TYPE DE CARTE SD	POUR CARTE SD	POUR CARTE MICRO SD (SDHC)
		

Aperçu du module		
Communication	SPI	SPI
Tension d'alimentation	4,5 à 5,5 (régulateur 5V intégré)	4,5 à 5,5 (régulateur 5V intégré)
Tension des signaux de communication	3,3 V uniquement	3,3 ou 5V (avec convertisseur de niveaux intégré)
Trous de fixation	2 x Ø2,2 (pour vis M2)	4 x Ø2,2 (pour vis M2)
Courants indicatifs	n.c.	0,2 mA (min) / 80mA (typique) / 200mA (max)
Lien (description/prix)	Lecteur Carte SD <a href="#">↗</a>	Lecteur Micro SD <a href="#">↗</a>

Sans hésiter la moindre seconde, je vous recommande fortement de **prendre un modèle intégrant un « convertisseur de niveaux intégré », comme présent dans le module « Micro SD » ci-dessus**, si vous débutez. Car ainsi, vous ne prendrez pas le risque d'endommager votre lecteur de SD card, si vous travaillez avec un Arduino communiquant sur des niveaux 0/5V (comme l'arduino Uno, Nano, ou Mega, par exemple). Au passage, faites très attention aux vendeurs qui vous disent que leurs modules sont « tolérant 5V ». Car derrière cela peut se cacher des abus de langage. Et même si en pratique la plupart des lecteurs de carte SD tiennent « le choc » lorsque alimentés en 5V au niveau de leurs broches de communication, leur durée de vie, quant à elle, ne peut être longtemps garantie dans ces conditions. C'est pourquoi **je vous recommande plutôt l'achat de lecteur de SD card intégrant un convertisseur 3V3 <-> 5V intégré, afin d'éviter tout soucis**. Voici d'ailleurs comment cela peut se formaliser, sur un lecteur de carte Micro SD :





*Nota : ce convertisseur ne s'occupe que des signaux de communication logiques (car l'alimentation, quant à elle, se fait bel et bien en 5 volts, typiquement)*



PassionElectronique.fr

À regarder de près, vous constaterez d'ailleurs la présence des broches « classiques » utilisées pour de la **communication par bus SPI, à savoir : les pins SCK, MISO, MOSI, et CS**. C'est donc idéal pour venir se raccorder sur un Arduino. Et c'est d'ailleurs tout aussi facile d'utilisation, avec la librairie SD.h, déjà intégrée à l'IDE Arduino.

À présent, concernant l'alimentation électrique de ces lecteurs de carte SD, **ceux-ci peuvent généralement s'alimenter aussi bien en 3,3 volts, qu'en 5 volts** (car ils sont la plupart du temps munis de régulateur interne 3V3 pour abaisser la tension 5V fournie sur leur VCC).

Au passage, privilégiez les lecteurs de carte SD dotés de trous de fixation dans les angles. Ainsi, il vous sera plus facile de les fixer « solidement » dans vos montages, sans que ça se balade partout ! Sinon, gare aux cartes SD qui s'abîment prématurément, ou qui se délogent !

Ah oui... tant que j'y pense, il y a encore deux choses à savoir, avant d'aller plus loin :

- Les cartes SD que vous prévoyez d'utiliser doivent au préalable être formatées (au format FAT16 ou FAT32, et de préférence au format FAT16, aussi nommé « FAT » tout court, si elle fait 4 GB ou moins)
- Les noms de fichiers ou répertoires que vous pourrez créer dessus devront respecter le « format 8.3 », c'est à dire comportant jusqu'à 8 caractères maxi pour le nom, et jusqu'à 3 caractères max pour l'extension (exemple de nom de fichier : 12345678.txt). Pour en savoir plus, n'hésitez pas à jeter un coup d'œil à la convention de nommage de fichier au format 8.3 ([https://en.wikipedia.org/wiki/8.3\\_filename](https://en.wikipedia.org/wiki/8.3_filename), en anglais), si vous souhaitez creuser la question 😊

Du reste, concernant la capacité des cartes SD utilisables avec un Arduino, je n'ai personnellement pas testé au-delà de 4 Go. Mais on peut bien évidemment aller au-delà, jusqu'à 32 Go, sauf erreur de ma part !

Important : assurez-vous toujours de bien prendre une carte SD compatible avec votre lecteur de SD card. Cela peut sembler évident dit comme ça, mais il faut savoir que, par exemple, un lecteur pour carte Micro SD n'est pas forcément automatiquement compatible avec les Micro SDHC, si non spécifié. Idem pour les

tailles de cartes supportées (8 GB, 32 GB, ...). C'est un point de détail lorsqu'on débute, mais qui a toute son importance. Donc mieux vaut s'assurer que tout soit bien compatible avec ce que vous prévoyez de mettre en œuvre, afin d'éviter toute déconvenue !

## 2. Branchement et câblage d'une micro SD card à un Arduino Uno, Nano, ou Mega (port SPI)

**Brancher un lecteur de carte SD sur un Arduino Uno, Nano, ou Mega** n'a rien de compliqué en soit. Car si on met de côté l'alimentation, les seuls fils à raccorder sont ceux du port SPI, à savoir : les bornes SCK, MISO, MOSI, et SS.

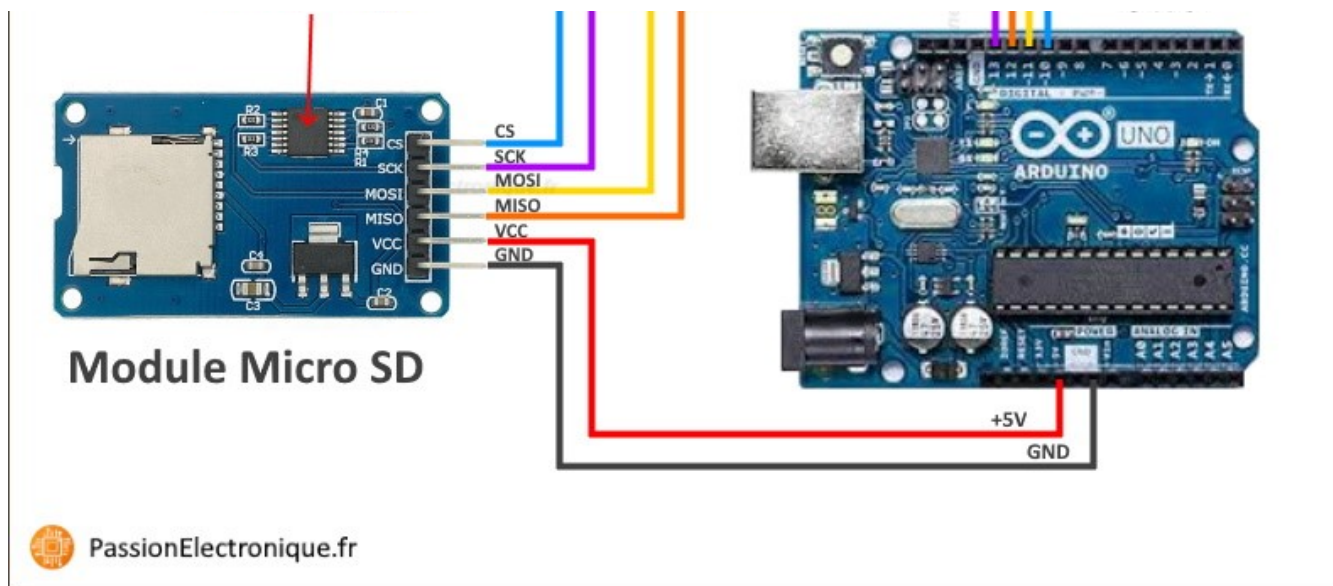
Bien entendu, suivant si vous êtes sur un Arduino de type Uno, Nano, Mega, ou autre, les bornes du port SPI peuvent varier. Ainsi, voici où **raccorder chacune des broches de votre lecteur de carte SD ou Micro SD, sur votre arduino** :

MODULE LECTEUR SD (SPI)	ARDUINO UNO	ARDUINO NANO	ARDUINO MEGA
<b>Broche CS</b>	D10	D10	53
<b>Broche SCK</b>	D13	D13	52
<b>Broche MOSI</b>	D11	D11	51
<b>Broche MISO</b>	D12	D12	50
<b>Broche VCC</b>	+5V	+5V	+5V
<b>Broche GND</b>	GND	GND	GND

Schématiquement, voici ce que cela peut donner avec un Arduino Uno (c'est d'ailleurs le montage que j'ai fait de mon côté, pour mes essais) :







Pour rappel : si jamais votre lecteur de SD CARD n'est pas équipé de convertisseur 3V3 / 5V intégré, il vous faudra vous même faire la conversion des niveaux de signaux, au niveau du port SPI, avec des transistors mosfets ou un circuit spécialisé externe (mais c'est fastidieux, et pas vraiment rentable !). Et si j'insiste autant sur ce point, c'est parce que **même si certains lecteurs de carte SD sont « dits » tolérant au 5V, au niveau de leurs broches de communication, cela pourrait à court ou moyen terme conduire à leur destruction**. Du coup, faites vraiment attention lorsque vous faites vos achats. Ou comme moi : achetez un modèle qui intègre déjà la puce de conversion de niveau, car cela évite bien des mésaventures !

### 3. La librairie SD.h (bibliothèque arduino)

Il existe plusieurs **librairies permettant de dialoguer avec un lecteur de carte SD, depuis un arduino**. Autant certaines sont simples et facile à utiliser (mais limitées), autant d'autres sont bien plus complexes (et permettent de faire bien plus de choses). Si vous débutez, je vous propose de partir sur la **bibliothèque déjà intégrée « de base », dans l'IDE Arduino par défaut**. C'est d'ailleurs de celle-ci dont je vais vous parler ici, car c'est un bon compromis entre stabilité, facilité d'utilisation, et fonctionnalités d'usage. D'ailleurs, il n'y aura aucune installation ou importation de librairie à faire, puisque nativement intégrée à l'environnement arduino.

Nous allons voir ici toutes les principales fonctions mises à notre disposition par cette librairie, selon moi, **pour lire, écrire, et récupérer des infos sur une carte SD**. À noter que tout cela est détaillé sur le site arduino, en anglais, pour ceux qui souhaitent davantage d'informations à ce sujet : <https://www.arduino.cc/en/reference/SD>.

Remarque : cette librairie est l'idéal pour débiter, selon moi. Par contre, dites-vous

que celle-ci n'est pas parfaite, et qu'elle souffre même de quelques lacunes (qui vous gêneront peut-être un peu, si vous cherchez à pousser les choses, un « cran plus loin ». Pour ma part, par exemple, je n'ai pas pu générer de noms de fichiers dynamiquement, sans avoir quelques warnings de compilation en retour. Toutefois, **cette bibliothèque reste d'après moi le meilleur compromis qui soit, entre simplicité et performance, pour tous ceux qui débutent avec Arduino** ! Bien sûr, si vous souhaitez faire des choses plus pointues ou plus spécifiques, il faudra peut-être utiliser une autre librairie que celle fournie nativement par l'IDE Arduino. Cela étant dit, vous verrez qu'on peut faire quand même pas mal de choses ici, avec la librairie SD.h, fournie dans l'environnement arduino 😊

### 3.1 - Initialisation de la librairie SD.h (fonction begin)

Afin de pouvoir utiliser la librairie SD.h, nous allons préalablement devoir appeler deux librairies dans notre code arduino :

- La librairie SPI.h (inclue de base dans l'IDE Arduino) : permettant d'assurer la communication entre l'arduino et le lecteur de carte SD, via le port SPI.
- La librairie SD.h (elle aussi nativement installée dans l'IDE Arduino) : permettant de mettre à disposition tout un tas de fonctions simples et efficaces pour lire, écrire, et interroger la carte SD. Pour rappel : il n'y a aucune installation ou importation à faire à ce niveau, car cette bibliothèque est déjà intégrée de base, dans l'environnement arduino.

Pour ce faire, il suffira d'écrire les lignes suivantes, dans l'entête de votre programme :

```
1 | #include <SPI.h>
2 | #include <SD.h>
```



Ensuite, il faudra bien évidemment initialiser la classe « SD », afin de pouvoir s'en servir. Au passage, **la bibliothèque SD.h nécessite qu'on lui précise sur quelle broche est raccordé le signal CS (ou SS), branché sur votre lecteur de carte SD**. Classiquement, celle-ci est câblée sur la pin D10 des arduino uno ou nano, ou pin 53 d'un arduino mega.

Au niveau du code arduino, voici comment cela peut s'écrire, si vous utilisez un Arduino de type Uno, avec la broche SS (CS) de votre lecteur de SD card branchée sur la pin D10 de votre

type `int`, avec la valeur de `CS` de votre module de SD sera branchée sur la pin `D10` de votre arduino (à mettre dans la fonction 'setup') :

```
1 | const int pinBranchementCS = 10;           // Le « 10 » indiquant ici que la br
2 | SD.begin(pinBranchementCS);
```

À noter que cette fonction `begin` retourne « `TRUE` » si l'initialisation s'est bien déroulée, ou « `FALSE` » dans le cas contraire. Vous verrez d'ailleurs comment exploiter ce code erreur en pratique, dans les exemples de codes présentés plus bas, dans cet article.

### 3.2 - Ouverture d'un accès en lecture ou écriture, vers un fichier sur la carte SD (fonction `open`)

**Afin de pouvoir lire ou écrire dans un fichier sur la carte SD, il faut au préalable exécuter la fonction « `open` ».** Et oui... cette fonction est commune pour faire ces deux choses là ! En fait :

- Si vous souhaitez lire un fichier, alors la fonction « `open` » vous permettra d'ouvrir un **accès en lecture** vers le fichier voulu
- Si vous souhaitez écrire dans un fichier, alors la fonction « `open` » vous permettra :
  - D'ouvrir un **accès en écriture** vers le fichier voulu, si celui-ci existe déjà
  - Ou de **créer le fichier puis ouvrir un accès en écriture** vers le fichier voulu, si celui-ci n'existait pas sur la carte SD

Comme vous l'aurez compris, la fonction « `open` » ne fait qu'ouvrir un canal vers le fichier visé. D'ailleurs, **suivant si vous essayez de lire ou écrire dans un fichier, la fonction « `open` » ne renverra pas forcément un code erreur suivant si le fichier ciblé existe ou pas.** En effet :

- Si vous essayez d'ouvrir un fichier en lecture, tout en sachant que celui-ci n'existe pas sur la carte SD, alors la fonction « `open` » vous renverra une erreur (retournera « `FALSE` », en fait)
- Mais si vous essayez d'ouvrir un fichier en écriture, tout en sachant que celui-ci n'existe pas, alors vous n'aurez aucun message d'erreur à proprement parler. Car un nouveau fichier « vide » sera simplement créé, avec le nom que vous aurez choisi. Et c'est seulement si jamais il était impossible au programme de créer ce nouveau fichier ou d'ouvrir le fichier existant, qu'un code erreur serait alors retourné.



J'espère que cette subtilité ne vous semblera pas trop obscure, car je ne saurais vous l'expliquer plus simplement !

Au final, l'appel de **la fonction « open » vous retournera** :

- Soit un objet de type File, pointant vers le fichier ouvert
- Soit une valeur booléenne égale à « FALSE », vous indiquant qu'il y a eu un problème lors de l'appel de cette fonction

Enfin, il faut savoir que **la fonction « open » accepte un paramètre optionnel, permettant de préciser le « mode de fonctionnement »**. Ce paramètre peut prendre 2 valeurs prédéfinies, qui sont :

- FILE\_READ, qui permet d'indiquer qu'on souhaite ouvrir le fichier en lecture seule, en lisant « depuis le début » (évident, me direz-vous !)
- FILE\_WRITE, qui permet d'indiquer qu'on souhaite ouvrir le fichier cible (en lecture ou écriture), mais « depuis la fin », cette fois-ci

Là encore, si ce n'est pas très clair, ne vous inquiétez pas. Car au final, voici les 2 lignes de code généralement utilisées en pratique :

- Si vous souhaitez ouvrir un fichier en lecture : **SD.open(nomDuFichier, FILE\_READ)**, ce qui vous permettra de lire le contenu de ce fichier depuis le début
- Si vous souhaitez ouvrir un fichier en écriture : **SD.open(nomDuFichier, FILE\_WRITE)**, ce qui vous permettra d'écrire dans ce fichier, à la suite des lignes déjà enregistrées dedans (car oui : cette fonction n'efface pas le contenu d'un fichier existant, mais permet en fait d'écrire « à la suite » des données déjà enregistrées dans ce fichier)

Le saviez-vous ? Il est possible d'entrer un nom de répertoire en même temps qu'un nom de fichier, afin de lire ou écrire dedans. Mais bien évidemment, ce répertoire doit déjà exister sur la carte SD. Car autant la fonction « open » peut créer un fichier si jamais celui-ci n'existe pas sur la SD card, autant cette fonction est incapable de faire de même avec un répertoire. Mais rassurez-vous ! Car il existe une solution pour cela, en appelant la fonction « mkdir », que nous verrons un peu plus loin !

### 3.3 - Fermer un fichier précédemment ouvert (fonction close)

Comme vous vous en doutez... une fois qu'on a ouvert un fichier... il faut le refermer ! Ou tout du moins, une fois qu'on en a fini avec lui ! Mais trait d'humour mis à part, **n'oubliez JAMAIS de fermer vos fichiers, une fois que vous en avez plus l'utilité**. Enfin... surtout lorsque ceux-ci sont ouverts en écriture, et nous allons voir pourquoi.

En effet, **c'est seulement au moment de l'appel de la fonction close (ou flush), que toutes les données sont enregistrées dans le fichier cible**. Par conséquent, ne pensez pas, en écrivant dans un fichier, que tout sera enregistré à la volée. D'ailleurs, histoire d'être parfaitement clair : dites-vous que si vous écrivez dans un fichier logé sur une carte SD, depuis un Arduino, et que vous n'exécutez pas la fonction « close », alors JAMAIS la moindre donnée ne sera réellement écrite ! Du coup : pensez toujours à fermer vos fichiers ouverts, une fois fini !

Au niveau du code, une ligne suffit, d'ailleurs :

```
1 | SD.close();
```



Vous constaterez que cette fonction ne retourne rien. Il n'est donc pas possible, à ce niveau, de savoir si le fichier a bien pu être fermé, ni si toutes les données ont bien pu être enregistrées (dans le cas d'un fichier ouvert en écriture, j'entends). Mais bon, rien n'est parfait 😊

### 3.4 - Vérifier si un fichier existe ou pas, ou un répertoire (fonction exists)

C'est vrai que je n'en ai pas parlé jusqu'à présent, mais il existe une fonction très intéressante, nommée « exists », permettant de **tester si un fichier ou un répertoire existe sur la carte SD**.

Cette fonction s'utilise très simplement, en lui donnant en paramètre :

- Soit le nom de fichier, dont on cherche à vérifier l'existence
- Soit le nom du répertoire, dont on cherche à vérifier l'existence
- Soit le nom d'un fichier incluant son ou ses répertoires, délimités par un slash (par exemple : « monrep1/monfic.txt »)

En retour, on obtient :

- « TRUE » si le fichier ou répertoire existe (est présent) sur la SD card
- « FALSE » dans le cas contraire

Ce genre de fonction est particulièrement intéressante :

- lorsqu'on cherche à effectuer un test d'existence avant toute ouverture de fichier en mode lecture (ça évite les erreurs d'accès en lecture, au beau milieu d'un programme informatique, par exemple !)
- lorsqu'on souhaite écrire dans un nouveau fichier, en effaçant l'ancien au préalable s'il existe déjà, par exemple (car pour rappel : la fonction open, lorsqu'appelée en écriture, n'efface rien d'un fichier existant, mais se place simplement à la fin de celui-ci, pour y rajouter les données à enregistrer)

### 3.5 - Effacer un fichier, présent dans une SD card (fonction remove)

Puisqu'on vient d'en parler, autant détailler la fonction « remove » sans plus attendre !

Car vous souhaiterez sûrement un jour **pouvoir effacer un fichier, ne serait-ce que pour en faire disparaître tout son contenu, et y stocker des données plus récentes à l'intérieur**, par exemple. Pour ce faire, rien de plus simple ! Car une seule ligne de code :

```
1 | SD.remove(nomDuFichierAsupprimer);           // Pour supprimer un fichier sur la
```

À noter que le nom de fichier à supprimer peut également contenir le chemin qui mène à lui, c'est-à-dire le ou les répertoires qui permettent d'y accéder. Dans ce cas, il vous suffira d'ajouter chacun des répertoires, avec un slash de délimitation entre chacun d'eux (par exemple : monrep1/sousrep2/monfic.txt).

Cette fonction « remove » retourne :

- la valeur « TRUE » si le fichier a bien pu être effacé
- la valeur « FALSE », si le fichier n'a pas pu être effacé
- ou aucune valeur, si le fichier n'existe pas

Remarque : il est très important de toujours vérifier la présence d'un fichier avant toute tentative d'effacement de celui-ci. D'ailleurs, on le voit bien ici, car la fonction « remove » ne retournera aucune valeur si le fichier n'existe pas sur la SD card. Du coup, en l'absence de retour d'une valeur égale à TRUE par cette fonction, vous ne sauriez dire si le problème vient d'un fichier absent, mal orthographié, ou d'un problème d'effacement sur la carte SD. D'où l'intérêt de toujours tout bien tester, avant de faire quoi que ce soit !

### 3.6 - Lire le contenu d'un fichier (fonction read)

Pour **lire le contenu d'un fichier présent sur une carte SD**, vous aurez besoin d'utiliser la fonction « read ». Mais vous aurez également besoin de réfléchir à comment vous souhaitez vous y prendre pour lire votre fichier. En effet, chaque fichier peut se lire d'octet en octet, tout comme ensemble d'octets après ensemble d'octets.

À savoir que :

- un pointeur permet de savoir à chaque instant à quel endroit on se situe dans le fichier (la position de celui-ci peut d'ailleurs être retournée par la fonction « position », si besoin)
- et qu'on peut parfaitement lire des octets depuis n'importe quel endroit d'un fichier, sans avoir à forcément tout lire depuis le début

Mais **généralement, on lit chaque fichier depuis le début, octet après octet, jusqu'à arriver à la fin**. Pour cela, on utilise :

- la fonction « read », donc, qui permet de lire 1 à plusieurs octets à la fois (et qui décale le pointeur d'autant d'octets lus, pour que la lecture suivante se fasse bien sur les octets suivants).
- la fonction « available », qui permet de savoir s'il reste ou non des octets à lire (sinon, cela signifie que soit le fichier était vide, soit on l'a lu jusqu'au bout)

Au niveau code de programmation, voici un exemple de ce qu'il est possible d'écrire, pour lire l'intégralité d'un fichier, du début à la fin :



```

1  File monFichierAlire = SD.open("monfichier.txt");    // Ouverture d'un accès en
2
3  if (monFichierAlire) {                               // Si l'ouverture s'est bi
4      while (dataFile.available()) {                   // Tant qu'il reste des oc
5          Serial.write(dataFile.read());               // ... on en lit un octet, p
6      }                                                 // (en sachant que le poin
7      dataFile.close();
8  } else {
9      Serial.println("Erreur lors de la tentative d'ouverture du fichier monfichier
10 }

```

En bref, c'est super simple ! Bien sûr, on peut pousser les choses un peu plus loin ici, et coupler cette fonction « read » avec d'autres fonctions, telles que :

- la fonction « size », qui permet de connaître la taille exacte d'un fichier (c'est-à-dire le nombre total d'octets à lire)
- la fonction « position », qui permet de donner la position courante du pointeur à l'intérieur du fichier (c'est-à-dire l'endroit où nous nous trouvons actuellement dans le fichier, qui, pour rappel, avance d'un cran après chaque lecture de type « read »)
- la fonction « seek », qui permet de « sauter » à un endroit précis à l'intérieur d'un fichier (cela permet de lire un ou plusieurs octets à un endroit précis, sans avoir à tout parcourir, depuis le début)
- et la fonction « peek » (à ne pas confondre avec « seek » !), qui elle permet de lire un octet dans un fichier (comme la fonction « read »), mais SANS faire avancer le pointeur de lecture, sur l'octet suivant à lire

### 3.7 - Écrire dans un fichier (fonction write)

Pour écrire dans un fichier sur une carte SD, il n'y a rien de vraiment bien compliqué ! Car une seule ligne permet de le faire, la plupart du temps. Au passage, vous pourrez **écrire un octet (caractère) à la fois, tout comme plusieurs à la fois**. En fait, il vous suffit d'écrire :

- SD.write(octetAecrire), pour écrire un octet dans le fichier préalablement ouvert, avec la fonction open
- SD.write(ensembleDoctets, longueur), pour écrire un ensemble d'octets (de type « byte array », ou « char array »)

**En retour, la fonction « write » renvoie le nombre d'octet qu'elle a pu écrire.** Par contre, ne



vous faites pas avoir ! Car, pour rappel, ces données ne sont réellement enregistrées qu'à l'appel de la fonction « close » (fermant le fichier), ou de la fonction « flush » (qui permet d'enregistrer tout ce qui était en attente d'enregistrement). Du coup, connaître le nombre d'octets écrits avec la fonction « write » n'est pas forcément quelque chose de pertinent, au sens où cela signifie simplement combien d'octets ont été écrits en mémoire tampon, en attente d'enregistrement « définitif ».

Sachez également qu'on retrouve 2 autres fonctions, semblables à la fonction « write » :

- La fonction « print », qui va plus loin que la fonction write, en permettant d'enregistrer des chiffres ou nombres au format texte ; d'ailleurs, on peut également préciser le format des nombres à enregistrer, suivant si celui-ci est au format binaire (BIN / base 2), octal (OCT, base 8), décimal (DEC, base 10), ou hexadécimal (HEX, base 16).
- La fonction « println », qui elle fait exactement la même chose que la fonction « print », mais en rajoutant un retour et saut à la ligne suivante en plus !

Remarque : la fonction « println » est très utile pour insérer en toute simplicité des sauts de lignes dans un fichier. Pour cela, il suffit d'écrire la commande suivante, à chaque fois que souhaité : « SD.println(); » (sans aucun argument, donc).

### 3.8 - Créer un répertoire (fonction mkdir)

Vous aurez peut-être également besoin de créer un répertoire, afin de classer tous vos différents fichiers sur carte SD. Pour cela, il existe une **fonction nommée « mkdir », qui permet de créer un répertoire avec le nom voulu** (mais attention : toujours en respectant le format 8.3, vu au début).

Cette fonction est particulièrement simple d'utilisation. Car là encore, une ligne suffit. Qui plus est, elle permet même de créer plusieurs sous-répertoires en même temps, si souhaité. Voici d'ailleurs deux exemples de syntaxe d'écriture, vous montrant comment créer un ou plusieurs répertoires à la fois :

- Pour créer un répertoire du nom de « monrep », il suffit par exemple d'écrire : **SD.mkdir(« monrep »);**
- Pour créer un répertoire « rep1 », avec un sous-répertoire « sousrepA » dedans, qui lui-même devra contenir un répertoire nommé « alpha », alors il suffit par exemple d'écrire : **SD.mkdir(« rep1/sousrepA/alpha »);** (cela va créer les 3 répertoires d'affilé)

En retour, la fonction « mkdir » renvoie :

- TRUE si la création du ou des répertoires s'est bien déroulée
- Ou FALSE, s'il y a eu un problème lors de la création du ou des répertoires

### 3.9 - Supprimer un répertoire (fonction rmdir)

À l'inverse de la fonction précédente, « **rmdir** » **permet de supprimer un répertoire ciblé, sur la carte SD**. Mais ATTENTION, car ce répertoire doit au préalable être vidé de tout son contenu (c'est à dire qu'il ne doit contenir ni fichier, ni répertoire, avant d'essayer de l'effacer). Sinon, cela ne fonctionnera pas.

Au niveau du code, cette fonction s'utilise en toute simplicité. Voici deux exemples d'appel :

- `SD.rmdir(« monrep »)` permet d'effacer un répertoire nommé « monrep »
- `SD.rmdir(« rep1/sousrepA/alpha »)` permet d'effacer un répertoire nommé « alpha », et uniquement lui (les autres répertoires devront être supprimés séparément, un à un, si souhaité)

Comme pour la fonction « mkdir », cette fonction retourne TRUE ou FALSE, selon si l'opération a pu se faire ou pas.

Voilà pour cette partie « librairie » ! À présent, nous allons passer à une partie bien plus fun, à savoir : la pratique ! Au programme : 4 exemples concrets, vous montrant « tout » ce qu'il est possible de faire avec cette bibliothèque, afin que vous puissiez faire vos premiers pas avec ! Après, il ne tiendra qu'à vous de vous exercer, de vous l'approprier, et d'approfondir tout ça 😊

## 4. Exemple de code #1 : récupérer les infos d'une carte SD (taille, format, ...)

Pour commencer, je vous propose de découvrir un exemple un peu en marge du reste, et qui n'attire pas de suite à la manipulation de fichiers sur carte SD. En fait, ce programme est juste une petite aparté, pour **apprendre à récupérer des infos de votre SD card depuis un Arduino**.

Comme vous vous en doutez, avant de se lancer dans le cœur du programme, il faut tout

Pour ma part, utilisant un arduino uno, voici le **schéma de raccordement que j'ai réalisé, pour câbler mon lecteur de carte Micro SD à mon Arduino Uno** :



```

14
15   Auteur :      Jérôme TOMSKI (https://passionelectronique.fr/)
16   Créé le :     07.06.2021
17
18   */
19   #include <SD.h>
20
21   // Variables SD Card
22   Sd2Card CarteSD;
23   SdVolume VolumeCarteSD;
24   uint32_t volumesize;
25
26   // Lecteur SD card branché sur les pins 10 (CS), 11 (MOSI), 12 (MISO), et 13 (S
27   #define sdCardSelect 10
28
29   // =====
30   // Initialisation programme
31   // =====
32   void setup () {
33
34       // Initialisation de la liaison série (pour retourner les infos au moniteur s
35       Serial.begin(9600);
36       Serial.println(F("Affichage des informations de la carte SD raccordée à l'Ard
37       Serial.println(F("=====
38       Serial.println();
39
40       // Étape 1 : test la présence d'une carte raccordée ou non
41       if (!CarteSD.init(SPI_HALF_SPEED, sdCardSelect)) {
42           Serial.println(F("Échec lors de l'initialisation. Essayez de jeter un coup
43           Serial.println(F("- est-ce que la carte SD est bien insérée dans le lecteur
44           Serial.println(F("- est-ce que vos branchements électriques sont corrects ?
45           Serial.println(F("- est-ce que le 'chipSelect' choisi dans le programme cor
46           Serial.println();
47           Serial.println(F("Appuyez sur RESET pour relancer le programme, au besoin."
48           while (1);          // Boucle infinie, arrêt du programme
49       } else {
50           Serial.println(F("Câblage correct, carte SD trouvée."));
51           Serial.println();
52       }
53
54       // Étape 2 : déterminer le type de carte SD insérée dans le lecteur
55       Serial.print(F("Type de carte SD insérée : "));
56       switch (CarteSD.type()) {
57           case SD_CARD_TYPE_SD1:
58               Serial.println(F("SD1"));
59               break;
60           case SD_CARD_TYPE_SD2:
61               Serial.println(F("SD2"));
62               break;
63           case SD_CARD_TYPE_SDHC:
64               Serial.println(F("SDHC"));
65               break;
66       }
67   }
68
69   // =====
70   // Initialisation programme
71   // =====
72   void loop () {
73       // =====
74       // Initialisation programme
75       // =====
76       // =====
77       // Initialisation programme
78       // =====
79       // =====
80       // Initialisation programme
81       // =====
82       // =====
83       // Initialisation programme
84       // =====
85       // =====
86       // Initialisation programme
87       // =====
88       // =====
89       // Initialisation programme
90       // =====
91       // =====
92       // Initialisation programme
93       // =====
94       // =====
95       // Initialisation programme
96       // =====
97       // =====
98       // Initialisation programme
99       // =====
100      // =====

```

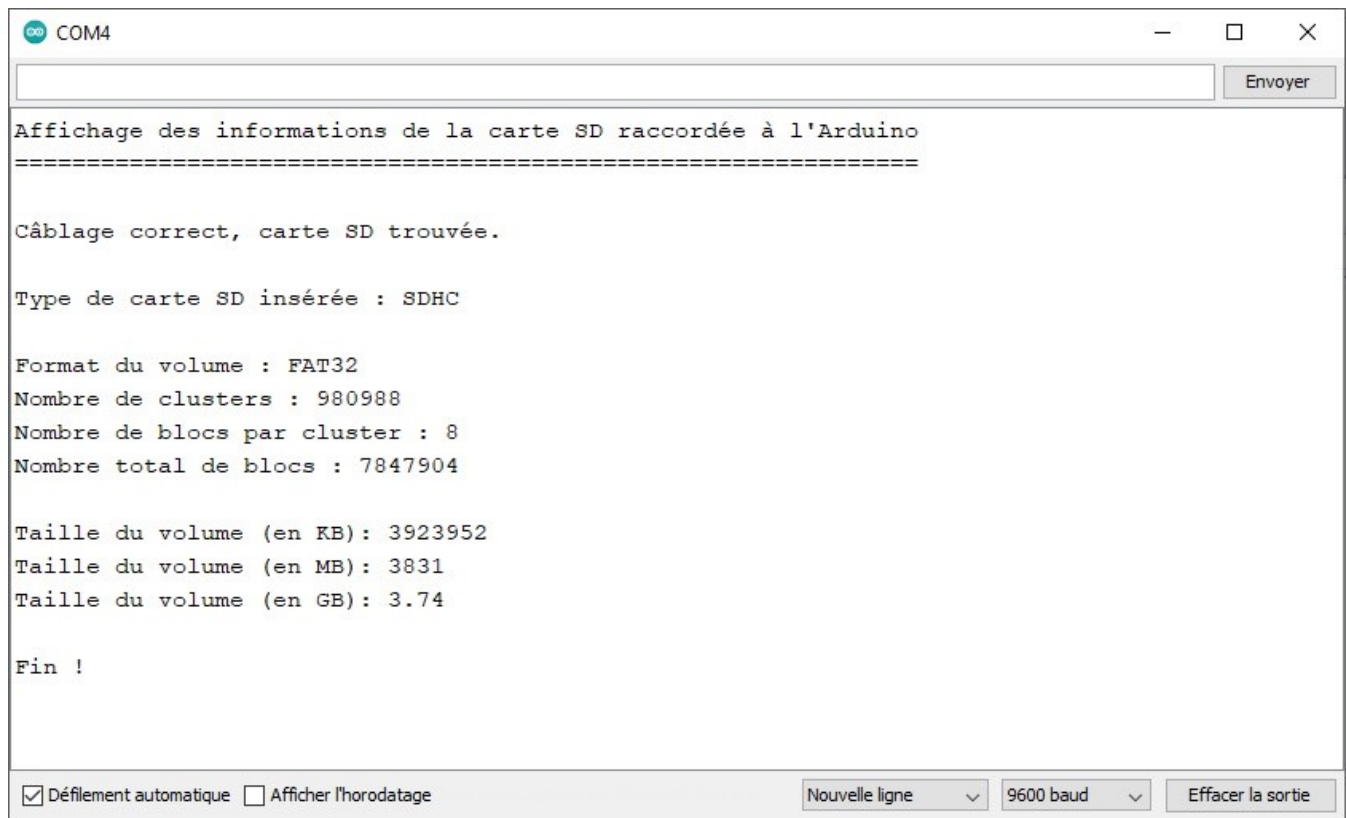
```
63     case SD_CARD_TYPE_SDHC:
64         Serial.println(F("SDHC"));
65         break;
66     default:
67         Serial.println(F("Inconnu"));
68 }
69 Serial.println();
70
71 // Étape 3 : tentative d'ouverture du volume/de la partition (qui doit être e
72 if (!VolumeCarteSD.init(CarteSD)) {
73     Serial.println(F("Aucune partition FAT16/FAT32 trouvée."));
74     Serial.println(F("Vérifiez si votre carte SD est bien formatée !"));
75     while (1); // Boucle infinie, arrêt du programme
76 } else {
77     Serial.print(F("Format du volume : FAT"));
78     Serial.println(VolumeCarteSD.fatType(), DEC);
79 }
80
81 // Étape 4 : affichage du nombre de cluster, et de blocs
82 Serial.print(F("Nombre de clusters : "));
83 Serial.println(VolumeCarteSD.clusterCount());
84 Serial.print(F("Nombre de blocs par cluster : "));
85 Serial.println(VolumeCarteSD.blocksPerCluster());
86 Serial.print(F("Nombre total de blocs : "));
87 Serial.println(VolumeCarteSD.blocksPerCluster() * VolumeCarteSD.clusterCount(
88 Serial.println();
89
90 // Étape 5 : affichage de la taille du volume, en KB/MB/GB
91 volumesize = VolumeCarteSD.clusterCount() * VolumeCarteSD.blocksPerCluster();
92 volumesize = volumesize / 2; // Nota : les blocs d'une carte SD font
93 Serial.print(F("Taille du volume (en KB): ")); Serial.println(volumesize);
94 Serial.print(F("Taille du volume (en MB): ")); Serial.println(volumesize / 10
95 Serial.print(F("Taille du volume (en GB): ")); Serial.println(volumesize / 10
96 Serial.println();
97
98 Serial.println(F("Fin !"));
99
100 }
101
102 // =====
103 // Boucle principale
104 // =====
105 void loop () {
106     // Vide (tout se passe dans la fonction "setup" !)
107 }
```

Vous noterez l'utilisation de fonctions « F » un peu partout dans le code, au niveau des



« Serial.print », afin d'économiser un maximum la mémoire RAM. Sans quoi... le programme pourrait devenir instable, du fait de manque d'espace suffisant.

Si tout est bien câblé, et le programme bien uploadé, vous devriez obtenir en réponse, sur le moniteur série de votre IDE Arduino, un écran semblable au mien (mais avec des valeurs propres à votre SD card, bien entendu !) :



```
COM4
Affichage des informations de la carte SD raccordée à l'Arduino
=====
Câblage correct, carte SD trouvée.
Type de carte SD insérée : SDHC
Format du volume : FAT32
Nombre de clusters : 980988
Nombre de blocs par cluster : 8
Nombre total de blocs : 7847904
Taille du volume (en KB): 3923952
Taille du volume (en MB): 3831
Taille du volume (en GB): 3.74
Fin !
```

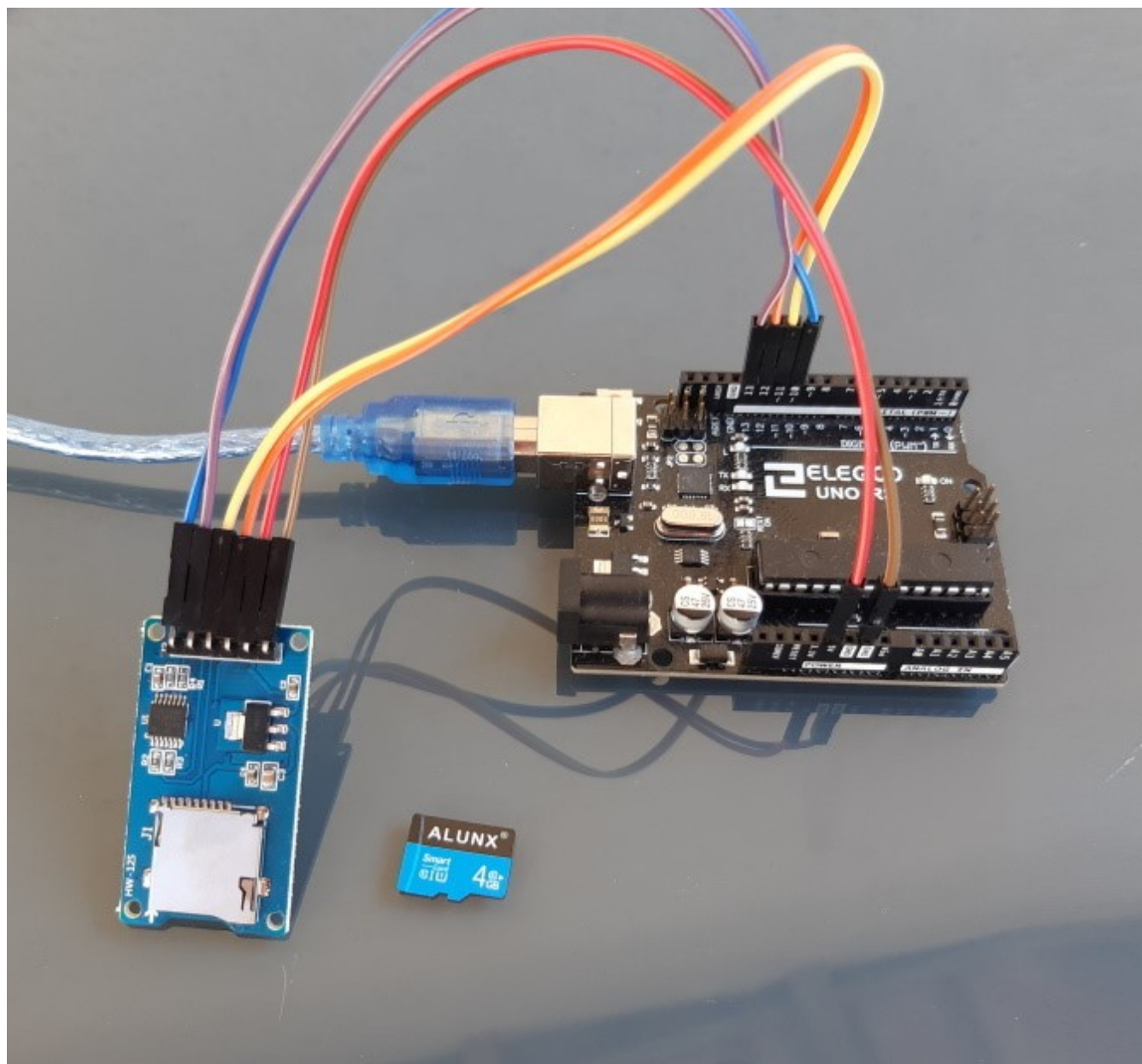
At the bottom of the window, there are checkboxes for "Défilement automatique" (checked) and "Afficher l'horodatage" (unchecked). On the right, there are dropdown menus for "Nouvelle ligne" and "9600 baud", and a button "Effacer la sortie".

Nota : c'est vrai que je n'en ai pas parlé jusqu'à présent, mais il faut bien évidemment que votre carte SD soit formatée, afin de pouvoir vous en servir (et attention, car l'acheter neuve ne garantit pas qu'elle le soit déjà). Les formats courants pour une carte SD arduino sont le FAT16 ou FAT32, avec une préférence pour le FAT16 lorsque c'est possible (détails disponibles ici en anglais, sur le site arduino : <https://www.arduino.cc/en/Reference/SDCardNotes>).

## 5. Exemple de code #2 : lire et écrire un fichier dans une SD card (read/write/dump)

Maintenant que nous avons fait nos premiers pas avec la librairie SD.h, nous allons voir ici **comment lire et écrire dans un fichier, sur une SD card**. Pour ce faire, je vous propose de réaliser le même montage que précédemment (cf. exemple 1), avec un lecteur de carte Micro SD branché sur un Arduino Uno. De mon côté, voici d'ailleurs un aperçu du montage réalisé :





Ici, nous allons réaliser une suite d'opérations, permettant de mettre en œuvre la plupart des fonctions les plus couramment utilisées. Ainsi, vous verrez comment se passent :

- L'initialisation de la communication avec un lecteur de SD card
- La suppression d'un fichier sur celle-ci, si déjà préexistant
- La création d'un fichier test, avec inscription d'une chaîne de caractère de test à l'intérieur
- Et la lecture de ce fichier (dump), avec l'affichage à l'écran de son contenu

En bref, de quoi faire un **tour d'horizon complet sur les opérations de lecture, écriture, et suppression de fichier sur carte SD, depuis un Arduino**. Pour y arriver, voici le programme que j'ai mis sur pied, pour réaliser cette série de tâches :

```

2
3
4
5
6
7
8
9
10   Fichier :      LectureEcritureCarteSD.ino
11
12   Description :   Ce programme permet de réaliser des opérations de lecture/écrit
13                   branchée sur un Arduino Uno (les résultats s'afficheront sur
14
15   Auteur :        Jérôme TOMSKI (https://passionelectronique.fr/)
16   Créé le :        09.06.2021
17
18   */
19   #include <SD.h>
20
21   // Variables utilisées pour la SD Card
22   #define sdCardPinChipSelect    10                                // Le lecteur
23   #define nomDuFichier            "fic-test.txt"                    // Format 8.3
24   #define texteDeTest            "Blabla écrit dans un fichier logé sur la carte S
25   File monFichier;
26
27   // =====
28   // Initialisation programme
29   // =====
30   void setup () {
31
32       // Initialisation de la liaison série (pour retourner les infos au moniteur s
33       Serial.begin(9600);
34       Serial.println(F("Programme de test de lecture/écriture de fichier sur Carte S
35       Serial.println(F("=====
36       Serial.println();
37
38       // -----
39       // Étape 1 : test si la carte SD est bien accessible depuis l'arduino
40       // -----
41       Serial.println(F("Étape 1 : Initialisation de la carte SD :"));
42       if (!SD.begin(sdCardPinChipSelect)) {
43           Serial.println(F("Échec de l'initialization !"));
44           while (1);    // Boucle infinie (stoppage du programme)
45       }
46       Serial.println(F("Initialisation terminée.));
47       Serial.println();
48
49       // -----
50       // Étape 2 : suppression du fichier 'nomDuFichier' si existe déjà

```

```
51 // -----
52 Serial.println(F("Étape 2 : suppression du fichier '" nomDuFichier "'", si déjà
53 if (!SD.exists(nomDuFichier)) {
54     Serial.println(F("Fichier '" nomDuFichier "'" inexistant."));
55 }
56 else {
57     Serial.println(F("Fichier '" nomDuFichier "'" déjà existant, donc lancement
58 // Suppression du fichier déjà existant
59 if(!SD.remove(nomDuFichier)) {
60     Serial.println(F("Échec de suppression du fichier '" nomDuFichier "'" prés
61 while (1);    // Boucle infinie (arrêt du programme)
62 }
63 else {
64     Serial.println(F("Suppression du fichier existant réussie."));
65 }
66 }
67 Serial.println();
68
69 // -----
70 // Étape 3 : écriture de données dans le fichier 'nomDuFichier'
71 // -----
72 Serial.println(F("Étape 3 : écriture de données dans le fichier '" nomDuFichier
73 monFichier = SD.open(nomDuFichier, FILE_WRITE);
74 if (monFichier) {
75     Serial.println(F("Écriture de la chaîne suivante, dans le fichier '" nomDuF
76     monFichier.println(texteDeTest);           // Écriture dans le fichier
77     monFichier.close();                       // Fermeture du fichier
78     Serial.println("Écriture terminée.");
79 }
80 else {
81     Serial.println(F("Échec d'ouverture en écriture, pour le fichier '" nomDuFi
82 while (1);    // Boucle infinie (arrêt du programme)
83 }
84 Serial.println();
85
86 // -----
87 // Étape 4 : lecture des données précédemment enregistrées
88 // -----
89 Serial.println(F("Étape 4 : lecture des données précédemment enregistrées dan
90 monFichier = SD.open(nomDuFichier, FILE_READ);
91 if (monFichier) {
92     Serial.println(F("Affichage du contenu du fichier '" nomDuFichier "'", ci-ap
93     Serial.write("-> Texte présent dans le fichier = ");
94     while (monFichier.available()) {           // Lecture, jusqu'à ce qu'il
95         Serial.write(monFichier.read());       // ... et affichage sur le mo
96     }
97     monFichier.close();                       // Fermeture du fichier
98     Serial.println(F("Lecture terminée "));
99
```

```
100 }
101 else {
102     Serial.println(F("Échec lors de l'ouverture en lecture du fichier '" nomDuF
103     while (1);    // Boucle infinie (arrêt du programme)
104 }
105 Serial.println();
106
107 // -----
108 // Fin du programme de test !
109 // -----
110 Serial.println(F("Fin !"));
111
112 }
113
114 // =====
115 // Boucle principale
116 // =====
117 void loop () {
118     // Vide, car tout se passe dans la fonction "setup" !
119 }
```

À noter que vous aurez une variante d'affichage entre la 1<sup>ère</sup> exécution de ce programme et les suivantes, du fait que le fichier de test sera absent la première fois, et présent les fois d'après. D'ailleurs, le programme vous indiquera clairement à l'écran si le fichier de test est déjà présent sur la carte SD ou pas, et l'effacera le cas échéant. Pour illustrer cela, voici deux aperçus écran que j'ai pu faire de mon côté, vous montrant la première puis la seconde exécution du même programme.

La première exécution :



Et la seconde fois (et fois d'après) :

Voilà ! À présent, nous allons corser un peu plus les choses, en intégrant un capteur de température et hygrométrie, pour faire des mesures à intervalles réguliers, tout en enregistrant celles-ci sur carte SD. Un Arduino Uno restera aux commandes, pour piloter tout ce beau monde ! Alors en avant !

## 6. Exemple de code #3 : enregistrer des mesures de température et hygrométrie avec un DHT22 (data logger)

Que diriez-vous si nous fabriquions un datalogger ? C'est-à-dire un petit **dispositif électronique, permettant d'enregistrer la température et le taux d'humidité ambiant, à intervalle régulier, tout en sauvegardant ces mesures au fur et à mesure, dans un fichier sur carte SD** ! Avec un Arduino à la commande, comme toujours !

Pour ce faire, nous allons partir du montage précédent, et rajouter un capteur de t°/hygrométrie de type DHT22 sur la broche D8 de notre Arduino (une autre broche digitale aurait pu convenir). Si besoin, au passage, n'hésitez à lire le [tutoriel sur le DHT22](#) que j'ai publié récemment, pour en apprendre plus sur ce capteur, à la fois simple, efficace, et pas cher 😊

Mais avant de nous jeter sur la partie codage, voyons ensemble le montage à réaliser :

Et pour donner vie à ce datalogger, voici le programme que j'ai écrit de mon côté :

[illegible]

```
29  const int typeDeDHT                = DHT22;                // Ici, le type de DHT
30  DHT dht(brocheDeBranchementDHT, typeDeDHT);
31
32  // Autres variables
33  const long delaiIntervalleDeMesures = 2000;                // Intervalle de mesure
34                                                                // ==> Nota : ne pas de
35
36  // =====
37  // Initialisation programme
38  // =====
39  void setup () {
40
41      // Initialisation de la liaison série (pour retourner les infos au moniteur s
42      Serial.begin(9600);
43      Serial.println(F("Programe DATALOGGER (enregistrement t°/hygro à partir d'un
44      Serial.println(F("=====
45      Serial.println();
46
47      // -----
48      // Vérification : est-ce que la carte SD est bien accessible depuis l'arduino
49      // -----
50      Serial.print(F("Initialisation de la carte SD..."));
51      if (!SD.begin(sdCardPinChipSelect)) {
52          Serial.println();
53          Serial.println();
54          Serial.println(F("Échec de l'initialisation du lecteur de SD card. Vérifiez
55          Serial.println(F("1. que la carte SD soit bien insérée"));
56          Serial.println(F("2. que votre câblage soit bon"));
57          Serial.println(F("3. que la variable 'sdCardPinChipSelect' corresponde bien
58          Serial.println(F("Et appuyez sur le bouton RESET de votre Arduino une fois
59          while (true);
60      }
61      Serial.println(F(" réussie !"));
62      Serial.println();
63
64      // -----
65      // Initialisation du DHT22
66      // -----
67      dht.begin();
68  }
69
70  // =====
71  // Boucle principale
72  // =====
73  void loop () {
74
75      // Lecture des données
76      float tauxHumidite = dht.readHumidity();                // Lecture du taux d'humidité
77      float temperature = dht.readTemperature();                // Lecture de la température
```

```

78
79 // Vérification si données bien reçues
80 if (isnan(tauxHumidite) || isnan(temperature)) {
81     Serial.println(F("Aucune valeur retournée par le DHT22. Est-il bien branché
82     delay(2000);
83     return;          // Si aucune valeur n'a été reçue par l'Arduino, on attend
84 }
85
86 // Mise en forme de ces données (un seul chiffre après la virgule)
87 String tauxHumiditeArrondi = String(tauxHumidite,1);    // Affichage d'une se
88 String temperatureArrondie = String(temperature,1);    // Affichage d'une se
89 tauxHumiditeArrondi.replace(".", ",");                // et on remplace les
90 temperatureArrondie.replace(".", ",");                // (permet par exempl
91
92 // Affichage des valeurs sur le moniteur série de l'IDE Arduino
93 Serial.print("Humidité = "); Serial.print(tauxHumiditeArrondi); Serial.print(
94 Serial.print("Température = "); Serial.print(temperatureArrondie); Serial.pri
95
96 // Enregistrement de ces données sur la carte SD
97 monFichier = SD.open(nomDuFichier, FILE_WRITE);
98 if (monFichier) {
99     monFichier.print(tauxHumiditeArrondi);
100    monFichier.print(";");                            // Délimiteur du fichier CSV
101    monFichier.println(temperatureArrondie);
102    monFichier.close();                                // L'enregistrement des données se
103    Serial.println(F("Enregistrement réussi en carte SD"));
104 }
105 else {
106     Serial.println(F("Erreur lors de la tentative d'ouverture du fichier en écr
107 }
108
109 // Temporisation de X secondes (2 sec min, pour que le DHT22 ait le temps de
110 Serial.println();
111 delay(delaiIntervalleDeMesures);
112 }

```

J'en profite d'ailleurs pour vous simuler différents cas possibles, suivant :

- si vous rencontrez un problème sur votre lecteur de carte SD
- si vous rencontrez un problème sur votre capteur de température et hygrométrie DHT22
- ou... si tout se passe bien !


Exemple de retour **si problème d'initialisation de la carte SD** :

Exemple de retour **si problème au niveau du capteur DHT22** :

Et **si tout se passe bien**, voici un exemple de retour sur le moniteur série de l'IDE Arduino :



Parallèlement, **vous pourrez retrouver toutes ces valeurs inscrites dans un fichier, sur votre carte SD**. Ce fichier porte le nom de « DONNEES.CSV » (tout simplement !), et toutes les mesures prises y seront stockées. Celles-ci seront formatées de la manière suivante : TAUX D'HUMIDITÉ ; TEMPÉRATURE (avec un point-virgule de séparation entre ces deux valeurs, et un saut/retour à la ligne pour passer aux valeurs suivantes). Voici d'ailleurs un extrait du fichier présent sur ma carte SD, après exécution du programme (à noter qu'il n'y a pas le moindre texte, mais seulement des données numériques à 1 décimale, et séparées entre-elles par un point-virgule) :



1	65,3;25,7
2	65,3;25,6
3	65,3;25,6
4	65,3;25,6
5	65,2;25,6
6	65,2;25,6
7	65,2;25,6
8	65,2;25,6
9	65,3;25,6
10	65,6;25,6
11	66,0;25,7
12	66,5;25,7
13	66,6;25,7
14	66,6;25,7
15	66,6;25,7
16	66,7;25,7
17	66,8;25,7
18	66,7;25,7
19	66,5;25,7
20	66,3;25,6
21	66,1;25,7

Ensuite, comme moi, vous pouvez **importer ces valeurs dans un tableur (de type Excel ou OpenOffice Calc), pour générer de beaux graphiques** ! D'ailleurs, voici ce que j'ai obtenu de mon côté, sur la base de données collectées au bout d'une petite heure de fonctionnement (à raison d'un échantillon de mesure toutes les deux secondes, ici) :

Bien sûr, il pourrait être bien plus sympa de faire des relevés sur une journée complète (24 heures, donc), avec un intervalle de mesure toutes les minutes, par exemple !

Remarque : au niveau du code de programmation présenté un peu plus haut, j'ai fait exprès ici de changer le format des différentes type de constantes. En effet, dans l'exemple 2, j'avais utilisé des « #define » pour définir le numéro de la pin CS et le nom du fichier test. Ici, dans l'exemple 3, j'ai utilisé les formats « const int » pour la pin CS, et « const char\* » pour le nom de fichier. Ceci est juste fait pour vous montrer qu'il y a différentes façons pour arriver au même résultat... mais encore faut-il les connaître 😊

## 7. Exemple de code #4 : génération d'un nom de fichier aléatoire, et enregistrement sur carte SD

Dernier exemple que j'ai tenu à vous partager ici : un **code permettant de générer un nom de fichier aléatoire, avec enregistrement sur carte SD**. Et si j'ai tenu à vous faire cela, c'est avant tout pour mettre en avant un « petit défaut » de la librairie SD.h, utilisée ici. En fait, ce défaut se situe au niveau du nom de fichier que l'on renseigne en utilisant la fonction « SD.open ». En effet :

- Si vous utilisez une constante (type #define ou const char\*) pour définir votre nom de fichier, alors vous n'aurez aucun souci
- Mais si utilisez une variable pour définir votre nom de fichier, alors vous aurez un petit message au moment de la compilation, sous la forme d'un avertissement. C'est à dire

[illegible]

```


23 char nomDuFichier[15],
24 File monFichier;
25
26 // =====
27 // Initialisation programme
28 // =====
29 void setup () {
30
31     // Initialisation de la liaison série (pour récupérer les infos en retour,
32     Serial.begin(9600);
33     Serial.println(F("Programme de test de génération de noms dynamiques, à enregi
34     Serial.println(F("=====
35     Serial.println();
36
37     randomSeed(analogRead(0)); // Préparation de la génération d'un nombre a
38
39     // -----
40     // Étape 1 : on teste si la carte SD est accessible depuis l'arduino
41     // -----
42     Serial.println(F("Étape 1 : Initialisation de la carte SD :"));
43     if (!SD.begin(sdCardPinChipSelect)) {
44         Serial.println(F("Échec de l'initialization !"));
45         while (1); // Boucle infinie (arrêt du programme)
46     }
47     Serial.println(F("Initialisation réussie.));
48     Serial.println();
49
50     // -----
51     // Étape 2 : génération d'un nom de fichier dynamique
52     // -----
53     Serial.println(F("Étape 2 : génération d'un nom de fichier aléatoire (de 1 à
54     long nombre8chifres = random(99999999); // Génère un nombre aléatoire
55     char baseFichier[8];
56     ltoa(nombre8chifres, baseFichier, 10); // Stocke le nombre décimal (
57
58     strcat(nomDuFichier, baseFichier); // Créer la base du nom de fi
59     strcat(nomDuFichier, ".txt"); // Puis ajoute une extension
60
61     Serial.print(F("Nom du fichier à créer = ")); // Et affiche ce nom sur le m
62     Serial.println(nomDuFichier);
63     Serial.println();
64
65     // -----
66     // Étape 3 : création du fichier sur la carte SD
67     // -----
68     Serial.println(F("Étape 3 : enregistrement d'une chaîne de caractères dans ce
69     monFichier = SD.open(nomDuFichier, FILE_WRITE); // Créé le fichier (à
70
71     if (monFichier) {

```

```
72     Serial.print(F("Ecriture d'une chaine de caracteres dans le fichier "));
73     Serial.print(nomDuFichier);
74     Serial.println(F(""));
75     monFichier.println("Ligne de test !");           // On écrit "Ligne de
76     monFichier.close();
77 } else {
78     Serial.print(F("-> Erreur lors de la tentative de création du fichier "));
79     Serial.print(nomDuFichier);
80     Serial.println(F(""));
81 }
82 Serial.println();
83
84 // -----
85 // Fin du programme de test !
86 // -----
87 Serial.println(F("Fin !"));
88
89 }
90
91 // =====
92 // Boucle principale
93 // =====
94 void loop () {
95     // Vide ici, car tout se passe dans la fonction "setup" !
96 }
```

Et comme je vous disais, voici ce que vous devriez voir apparaître en bas de votre IDE Arduino, après compilation du programme (des lignes en rouge, dans la partie basse de l'écran) :

Pour être plus précis, voici ce message d'erreur qui y figure, au format texte :



```
1 C:\Program Files\WindowsApps\ArduinoLLC.ArduinoIDE_1.8.49.0_x86__mdqgnx93n4wtt\  
2 C:\Program Files\WindowsApps\ArduinoLLC.ArduinoIDE_1.8.49.0_x86__mdqgnx93n4wtt\  
3     filepath += pathidx;  
4         ^  
5 C:\Program Files\WindowsApps\ArduinoLLC.ArduinoIDE_1.8.49.0_x86__mdqgnx93n4wtt\  
6     int pathidx;  
7         ^
```

Bien sûr, ce petit « bug » de librairie pourrait être corrigé à même la librairie, mais en cas de mise à jour de cette dernière, toutes ces modifications pourraient être perdues. Du coup, pour ma part, je laisse les choses telles quel, sachant qu'elles n'empêchent en rien la bonne exécution du programme ! Pour preuve : voici **ce que j'obtiens sur mon moniteur série (sachant que le nom du fichier généré change à chaque exécution, bien sûr, comme prévu !)**.

Bien évidemment, on aurait pu utiliser une autre librairie arduino ici, pour « contourner » ces warning à la compilation. Mais j'ai préféré rester sur des choses simples et fonctionnelles, pour les débutants, tout en sachant qu'on pourra ensuite pousser les choses un peu plus, une fois les bases maîtrisées !



## 8. Carte SD Arduino : conclusion !

Nous voici au terme de cet article sur **comment brancher un lecteur de carte SD sur un Arduino** ! J'espère que tout ceci vous aura plu, et que cela vous donnera envie d'intégrer des cartes SD à vos futurs montages ! J'espère également que tous les exemples que je vous ai concocté seront suffisamment clairs pour vous, afin de bien **comprendre les fonctions de base pour piloter une SD card depuis un Arduino**. Du reste, il ne s'agit évidemment là que d'une introduction aux SD card, et une base d'apprentissage pour tous ceux qui débutent. Il faudra bien évidemment compléter tout cela, et creuser au niveau d'autres librairies arduino plus poussées, si vous souhaitez aller bien plus loin !

Bon... quant à moi, il ne me reste plus qu'à vous souhaiter une bonne journée, et à vous dire : à la prochaine 😊

Jérôme.

À découvrir aussi : [comment lire ou écrire dans la mémoire EEPROM d'un Arduino ?](#)

Ce contenu vous plaît ? Alors abonnez-vous à la Newsletter pour ne rien louper !

 Recevoir la Newsletter !

### JEROME

Passionné par tout ce qui touche à l'électronique, sans toutefois être expert ni ingénieur, j'ai à coeur de partager ici avec vous toutes mes connaissances, réalisations, expériences, avis et découvertes ! Alors à

très vite !

(\*) Mis à jour le 11/07/2021

---

13 commentaires sur "Carte SD Arduino : branchement, librairie de base, et exemples de code (pour Arduino Uno, Nano, Mega)"