Les micro-contrôleurs présents sur les *Arduino* que nous utilisons au laboratoire, possèdent entre 2 et 4 ports d'entrées/sorties de 8 lignes (ou « pins ») chacun. ***Toutes les lignes peuvent être individuellement configurées en entrée ou en sortie***.

1. Étude logicielle

Il est possible de programmer ces micro-contrôleurs de 2 manières :

- ***Les fonctions Arduino*** : elles simplifient la programmation et permettent l'utilisation aisée d'extensions (écrans LCD, tactiles, claviers, dispositifs de transmission)
- ***Les fonctions AVR*** : on utilise alors directement les registres du micro-contrôleur pour le programmer (plus complexe mais aussi plus puissant)

Fonctions Arduino

Deux fonctions Arduino sont prévues pour configurer chaque pin, qui sont numérotées de 0 à 13 sur un Arduino Uno :

- `pinMode(<N°Pin>, <INPUT ou OUTPUT>)` : permet de configurer la pin en entrée ou en sortie
- `digitalWrite(<N°Pin>, <LOW ou HIGH>)` :
  - Si ***la pin est configurée en sortie***, cette commande permet de ***changer son état logiqu***e (0 ou 1, présence tension ou non)
  - Si ***la pin est configurée en entrée***, cette commande permet de ***changer sa configuration*** (pull down ou up, voir page suivante)

Par défaut, au démarrage du micro-contrôleur, toutes les pins sont configurées en entrées pull-down

Fonctions AVR

Deux registres de 8 bits sont affectés au fonctionnement de chaque port et une adresse permet d'accéder au port lorsque celui ci est configuré en entrée :

Un registre de direction DDR*n* :

- La ligne (ou PIN) est ***configurée en sortie*** si le bit correspondant du registre **DDR*n* est à 1**
- La ligne (ou PIN) est ***configurée en entrée*** si le bit correspondant du registre **DDR*n* est à 0**

Un registre de donnée PORTn :

- Si la ***ligne est configurée en entrée***, le bit correspondant du registre **PORT*n*** permet sa ***configuration*** (voir *Étude matérielle*)
- Si la ***ligne est configurée en sortie***, le bit correspondant du registre **PORT*n*** permet d'***écrire l'état*** souhaité de la ligne de sortie

Un registre de lecture d'état PINn :

- permet la ***lecture de l'état*** d'une ligne ***configurée en entrée***

Écrire ou lire une seule ligne (ou pin) impose donc de réaliser un « masque »

- Pour **configurer** la **pin « 0 »** du **PORTB** en **sortie** sans agir sur les autres pins du port, il faut réaliser un « OU logique » entre le PORTB et la valeur binaire « 0b00000001 », qui correspond à « 0x01 » en hexadécimal

```
DDRB = (DDRB || 0x01) ;
```
- En C/C++, l'opérateur « || » réalise un OU Logique

- Pour **lire l'état** la **pin « 7 »** du **PORTC** sans tenir compte de l'état des autres pins du port, il faut réaliser un « ET logique » entre le PORT1 et la valeur binaire « 0b10000000 », qui correspond à « 0x80 » en hexadécimal

```
etatEntree7 = (PORTC && 0x80) ;
```
- « && » est un ET logique ;

Application 1 :

Le port D du µC (pins Arduino numérotées de 0 à 7) est utilisé pour des entrées et des sorties : les 4 premières lignes (pins 0 à 3) sont configurées en entrée, les 4 suivantes en sortie

1. Arduino: Déterminer les 4 lignes de code nécessaire pour cette configuration
2. AVR : Déterminer, en décimal et hexadécimal, la valeur à donner au registre DDRD :

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Décimal	Hexa
DDRD										

3. Arduino : Donner les commandes permettant d'écrire l'état logique « 1 » sur chaque sortie

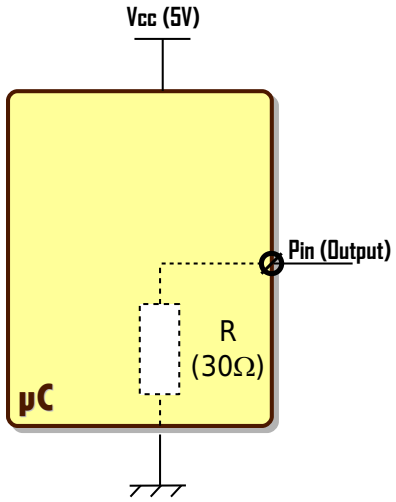
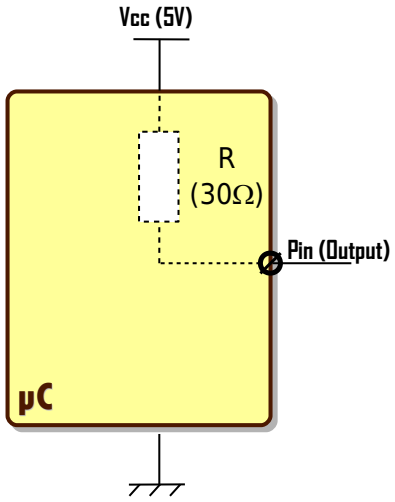
2. Étude matérielle

Pin configurée en sortie

Les pins configurées en sortie (ou *output*) sont dites à ***basse impédance*** : leur résistance interne est faible (**environ 30 Ω**)

- Elles acceptent un ***courant max I=40mA***, ce qui permet d'alimenter directement plusieurs LED.
- ***Si le courant nécessaire dépasse 20mA, il est préférable qu'il soit fourni par une source externe, car la tension chute alors à 4,4V*** *(voir l'extrait de la documentation Atmel en annexe)*

Modèles électriques

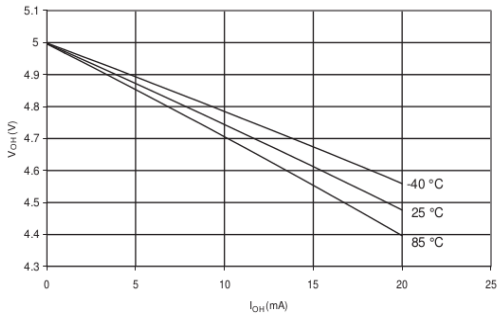
	Pin de sortie à l'état *« Low »*	Pin de sortie à l'état *« High »*
Commandes relatives	`pinMode(0, OUTPUT)` `digitalWrite(0, LOW)`	`pinMode(0, OUTPUT)` `digitalWrite(0, HIGH)`
Modèle électrique		

28.1 Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except RESET with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on RESET with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Figure 29-162. ATmega328P: I/O Pin Output Voltage vs. Source Current(V_{CC} = 5 V)



Extrait de la documentation de l'ATmega328P (Arduino Uno) concernant les sorties

Pin configurée en entrée

Les pins configurées en entrée sont à ***haute impédance*** : leur résistance interne est grande (**environ 20 kΩ**).
2 modes sont possibles :

- `digitalWrite(<pin>, LOW)` ou ***valeur du registre DDR à 0*** : l'entrée est ***reliée à la masse***. On appelle ce mode **PULL-DOWN**
- `digitalWrite(<pin>, HIGH)` ou ***valeur du registre DDR à 1*** : l'entrée est ***reliée à la tension d'alimentation Vcc*** (5V). On appelle ce mode **PULL-UP**

En mode PULL-UP , il est possible de connecter directement des contacts (ou switchs) reliées à la masse.

L'entrée se comporte comme un voltmètre qui renvoie 2 états discrets (0 ou 1). Il est possible de lire cet état grâce à la fonction arduino `digitalRead(<pin>)`, ou par la lecture du registre PINx, x étant le port considéré (A, B, C ou D)

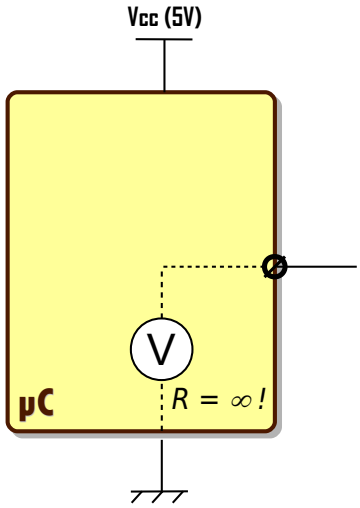
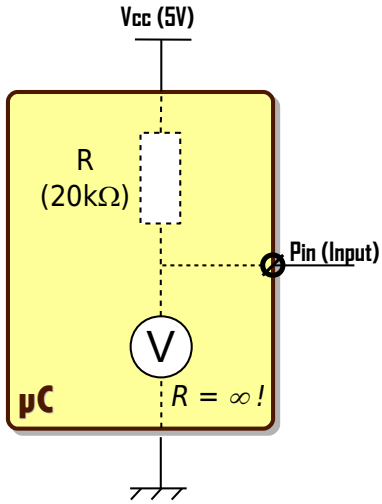
- Si la ***tension sur l'entrée est comprise entre -0,5 V et 1V, la valeur renvoyée est « 0 »***
- Si la ***tension sur l'entrée est comprise entre 3V et 5,5V, la valeur renvoyée est « 1 »***
- ***Dans tous les autres cas, la valeur est indéfinie : il est donc primordial de toujours fixer la tension sur la pin, elle doit donc toujours être connectée soit à la masse, soit à l'alimentation, mais jamais laissée « en l'air ».***

T_A = -40°C to 85°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
V_{IL}	Input Low Voltage, except XTAL1 and RESET pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	-0.5 -0.5		$0.2V_{CC}$ (1) $0.3V_{CC}$ (1)	V
V_{IH}	Input High Voltage, except XTAL1 and RESET pins	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.7V_{CC}$ (2) $0.6V_{CC}$ (2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IL1}	Input Low Voltage, XTAL1 pin	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$ (1)	V
V_{IH1}	Input High Voltage, XTAL1 pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.8V_{CC}$ (2) $0.7V_{CC}$ (2)		V_{CC} + 0.5 V_{CC} + 0.5	V

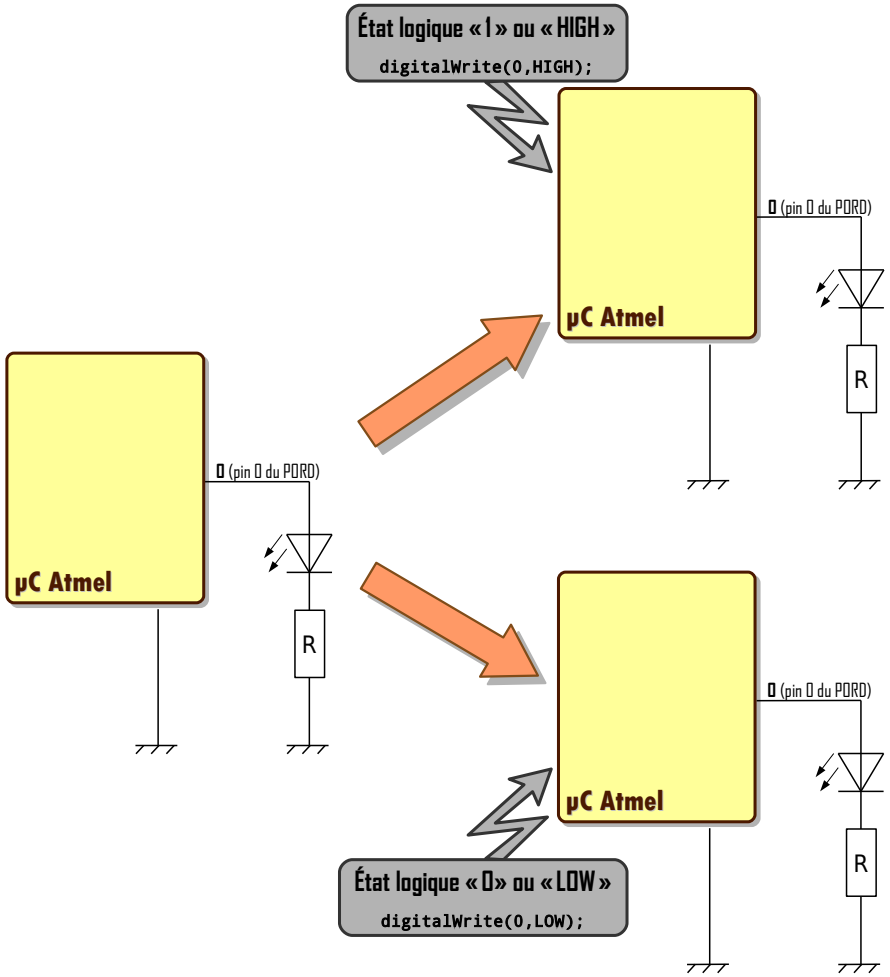
Extrait de la documentation de l'ATmega328P (Arduino Uno) concernant les entrées

Modèles électriques

	Pin d'entrée en *PULL-DOWN*	Pin d'entrée en *PULL-UP*
Commandes relatives	`pinMode(0, INPUT)` `digitalWrite(0, LOW)`	`pinMode(0, INPUT)` `digitalWrite(0, HIGH)`
Modèle électrique		

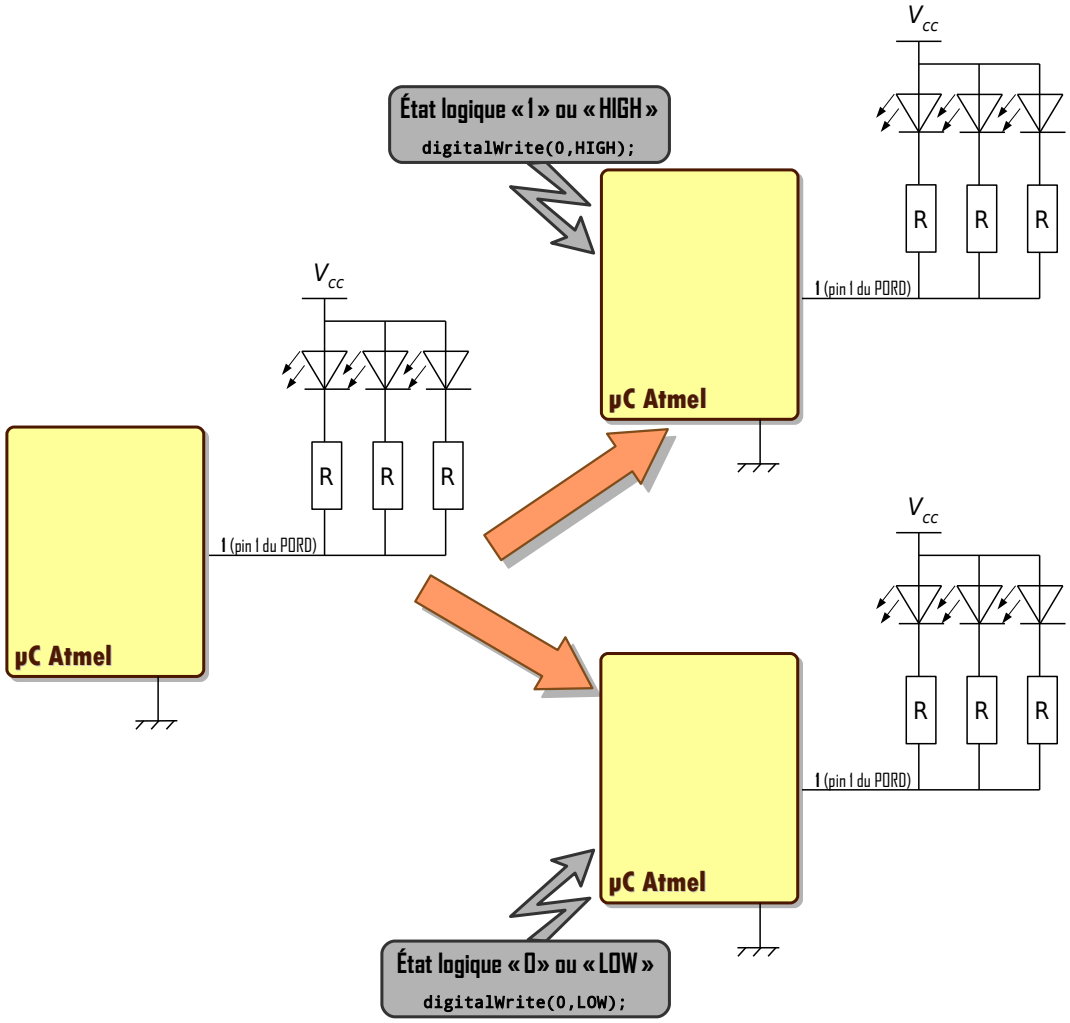
3. APPLICATIONS

Application 2 : utilisation d'une sortie pour commander l'état d'une LED



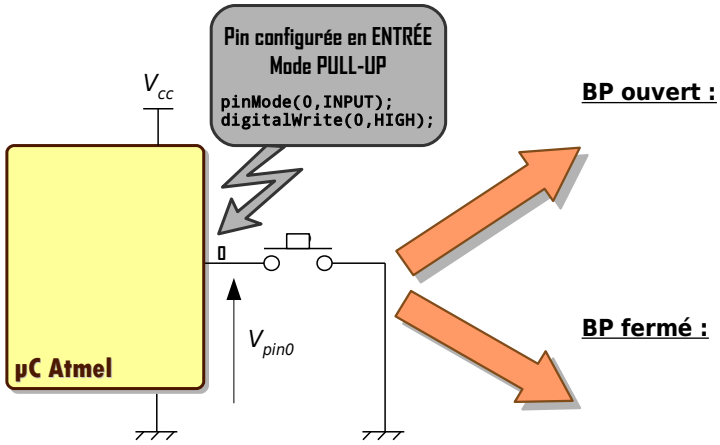
1. Reproduire pour chacun des cas le modèle électrique interne de la pin selon le tableau précédent.
2. Dessiner pour chaque cas le courant dans le circuit
3. Dessiner pour chaque cas les tensions dans le circuit (Vcc, Vd, Vr)
4. Déterminer pour chaque cas si la valeur de Vd peut atteindre 1 V (condition pour que la LED s'allume)
5. Déterminer la valeur de R pour limiter le courant de la LED à I = 10 mA

*Application 3 : utilisation d'une **sortie** pour commander l'état de plusieurs LEDs (l'alimentation est externe)*



1. Reproduire pour chacun des cas le modèle électrique interne de la pin selon le tableau précédent.
2. Dessiner pour chaque cas les courants et les tensions du circuit
3. Déterminer pour chaque cas si la valeur de Vd peut atteindre 1 V (condition pour que la LED s'allume)
4. Déterminer la valeur du courant traversant la pin du micro-contrôleur.
5. Le micro-contrôleur fournit il ce courant ? Justifier l'usage de ce montage en rapport avec le précédent lorsqu'on souhaite alimenter plusieurs LEDs.
6. Vérifier que le courant ne dépasse pas les limites physiques de la pin (voir tableau « absolute ratings »)

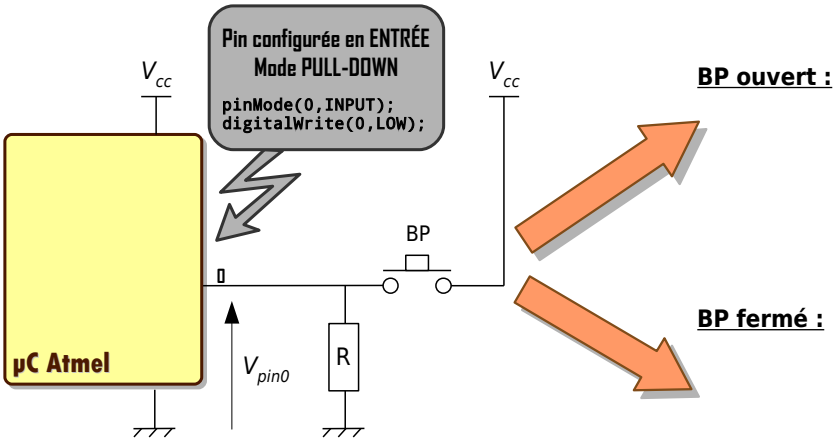
*Application 4 : utilisation d'une **entrée** en mode PULL-UP pour lire l'état d'un switch*



1. Reproduire le modèle électrique interne de la pin selon le tableau précédent.

2. **Déterminer la valeur de la tension V_{pin0} dans les 2 cas (switch ouvert et fermé), en déduire la valeur retournée par la fonction digitalRead(0)**
3. **Déterminer la validité du montage dans le cas d'une entrée configurée en mode PULL-DOWN**

Application 4 : utilisation d'une entrée en mode PULL-DOWN pour lire l'état d'un switch (R=10kΩ)

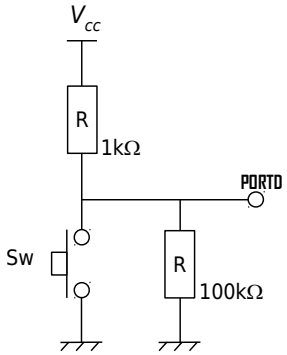


1. **Reproduire le modèle électrique interne de la pin selon le tableau précédant.**
2. **Déterminer la valeur de la tension V_{pin0} dans les 2 cas (switch ouvert et fermé), en déduire à la valeur retournée par la fonction digitalRead(0)**
3. **Que se passerait il sans la résistance R reliée à la masse ?**
4. **Déterminer la validité du montage dans le cas d'une entrée configurée en mode PULL-UP**

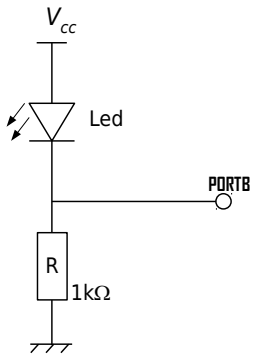
4. Application sur la platine STK200

Dans le cas de la platine STK200 utilisée au laboratoire :

- Les ***lignes du port D, numérotées de 0 à 7***, et ***prévues pour être des entrées***, sont connectées aux contacts selon le montage suivant :



- Les ***lignes du port B, prévues pour être des sorties***, sont reliées aux LEDs selon le montage suivant :



Vous devez écrire un programme C++ Arduino réalisant une fonction mémoire à arrêt prioritaire :

- Marche correspond à la pin 7 du port D (pin Arduino N°7)
- Arrêt à la pin 0 du port D (pin Arduino N°0)
- La LED utilisée pour présenter l'état de la mémoire sera la pin 0 du port B (pin Arduino N°8)

1. Pour un mode de configuration PULL-DOWN, définir la valeur de la tension aux bornes de l'entrée puis le résultat de la fonction `digitalRead(7)` lorsque le le Bp est ouvert
2. Pour un mode de configuration PULL-DOWN, définir la valeur de la tension aux bornes de l'entrée puis le résultat de la fonction `digitalRead(7)` lorsque le le Bp est fermé
3. Mêmes questions que les 2 précédentes pour un mode de configuration PULL-UP
4. En déduire le mode de configuration des entrées (PULL-UP/PULL-DOWN) qui fonctionne dans le cas du montage proposé
5. Définir l'état (allumée ou éteinte) de la LED reliée à la pin Arduino N°8 lorsque son état est 1 (`digitalWrite(8,HIGH)`)
6. Définir l'état (allumée ou éteinte) de la LED reliée à la pin Arduino N°8 lorsque son état est 0 (`digitalWrite(8,LOW)`)
7. Réaliser le programme Arduino pour notre fonction mémoire en tenant compte du résultat précédent :
   - La fonction `void setup()` comportera la configuration des pins (entrée, sortie, pull-up, pull-down)
   - La fonction `void loop()` comportera la fonction mémoire
   - Pour simplifier l'écriture :
     - `if (digitalRead(<pin>)==1)` peut s'écrire `if (digitalRead(<pin>))`
     - `if (digitalRead(<pin>)==0)` peut s'écrire `if (not digitalRead(<pin>))` ou `if ( ! digitalRead(<pin>))`