

# subystème

- [la base des subsystemes](#)
- [moteur](#)
- [Digital input](#)
- [Piston](#)

# la base des subsystemes

quand on créer un subsysteme le programme de base est celui-ci:

```
// Copyright (c) FIRST and other WPILib contributors.  
// Open Source Software; you can modify and/or share it under the terms of  
// the WPILib BSD license file in the root directory of this project.
```

```
package frc.robot.subsystems;  
  
import edu.wpi.first.wpilibj2.command.SubsystemBase;  
  
public class Bougie extends SubsystemBase {  
    /** Creates a new Bougie. */  
    public Bougie() {}  
    }  
    @Override  
    public void periodic() {  
        // This method will be called once per scheduler run  
    }  
}
```

la section import est l'endroit ou tu mets tes codes préfait

dans le public class Bougie extends SubsystemBase {} tu met tes moteur que tu créé et en dessous de public Bougie() {} tu mets tes fonction(une fonction voir plus bas)

dans le périodic tu mes une action qui ce fera en permanance. C'est une boucle

## une fonction

une fonction est une action préfait qui fera tous ce que tu a coder a l'interieur chaque fois que tu l'utilise dans une commande

exmple de fonction:

```
public void balaye(double vitesse){  
    balaye.set(vitesse);  
}
```

chaque fois que tu veux utiliser un moteur par exemple tu appelles la fonction. dans cette exemple on lui dit de tourner et ensuite on définie ça vitesse en %.

# moteur

## intro:

Quand on utilise un moteur ce qu'on code est un controleur de moteur. On en utilise 4 différent en robotique: CANspark max, CANspark flex, Talon SRX, Talon FX.

CANspark max= moteur NEO

CANspark flex = moteur vortex

Talon FX = moteur Kraken

Talon SRX = autre moteur comme 775 pro ou SIM

## pour créer l'objet du moteur:

les 2 spark ce ressemble beaucoup.

exemple:

```
final SparkFlex lance = new SparkFlex(17 , MotorType.kBrushless);  
final SparkMax accumulateur = new SparkMax(18,  
MotorType.kBrushless);
```

final signifie qu'on ne peut modifier cette ligne a d'autre endroit dans le code

SparkFlex ou SparkMax le controleur a utiliser

lance est le nom du moteur c'est toi qui décide du nom

new SparkFlex signifie que tu veux en créer un nouveau

le 17 est l'ID c'est le nombre qui identifie ton moteur

MotorType.kBrushless ou MotorType.kBrush signifie que le moteur est soit avec ou sans broche.  
Les moteur a la robotique sont tous brusless

les 2 Talon ce ressemble beaucoup.

exemple:

```
final WPI_TalonSRX pince = new WPI_TalonSRX(2);
```

WPI\_TalonSRX ou WPI\_TalonFX le controleur a utiliser

pince est le nom du moteur

new WPI\_TalonSRX signifie que tu veux en créer un nouveau

le 2 est l'ID

ensuite pour dire quel utiliter donner au moteur on créé une fonction.

exemple:

```
public void lance(double vitesse){  
    Lanceur.set(vitesse);  
}
```

public signifie qu'il peut être utilisé partout dans le code

void signifie que tu retourne aucune valeur

le void pourrait être renplacer par:

double= renvoie un chiffre a virgule, int= renvoie un chiffre plein, boolean renvoie soit true ou false  
pour des objet a seulement 2 état par exemple,String= renvoie un message.

lance est le nom de la fonction elle sera utilisé dans les command(voir le chapitre sur les  
Commande)

double vitesse est une variable qui pourra être modifier dans une commande

lanceur est le nom du moteur utiliser dans la fonction

set est ce que le moteur fait dans la fonction. Set mets la vitesse a la valeur mis dans les parentèse  
en %

un fonction peut servir pour différente chose mais on l'utilise principalement pour: définir la  
vitesse d'un moteur( comme l'exemple), prendre la valeur d'un encodeur(voir juste en bas, ou  
définir la position d'un encodeur(voir plus loin).

```
public double encodeur(){  
    return rotatione.getEncoder().getPosition();  
}
```

public, double,encodeur et sont des mot qui on été défini plus haut

return signifie que ce qu'il y a après c'est ce que tu retournes comme valeur

rotatione.getEncoder().getPosition() est que tu prends la position de l'encodeur du moteur rotatione pour définir la position d'un encodeur pour par exemple reset l'encodeur

```
public void reset(){  
    monte.getEncoder().setPosition(0);  
}
```

à la place de rotatione.getEncoder().getPosition() on met monte.getEncoder().setPosition() et le 0 est la valeur de l'encodeur qu'on définit

# Digital input

## Intro:

un digital input est quelque chose qui renvoie un signal on/off. Il y en a plusieurs qu'on utilise

une limit Switch est un module avec un bouton qui envoie un signal s'il est activé

une photocell est un laser qui est réfléchi vers un détecteur qui renvoie un signal si le détecteur ne détecte plus le laser donc que quelque chose est passé devant

## pour créer l'objet d'un digital input:

```
final DigitalInput limit3 = new DigitalInput(1);
```

limit3 est le nom donné à la limit switch

le 1 est le nombre du port DIO dans le Roborio (C'est comme le cœur du robot)

## pour une fonction:

```
public boolean enHaut(){  
    return limit3.get();  
}
```

boolean signifie que tu renvoie true/false (on/off)

enHaut est le nom de la fonction

return signifie que tu retournes une information

limit3 est le nom du digital input

get() signifie que tu prends l'état de la limit switch (on ou off)

# Piston

pour programmer un piston on programme les valves ou solenoid. Il peut être double ou simple.

## pour créé l'objet du DoubleSolenoid

```
private DoubleSolenoid brake = new DoubleSolenoid(PneumaticsModuleType.CTREPCM, Constants.brakeouvre, Constants.brakeferme);
```

le private veux dire que il ne peut être utilisé a une autre place dans le code

DoubleSolenoid est pour le piston

brake est le nom du piston

new DoubleSolenoid est pour dire que tu veux en créé un nouveau

PneumaticsModuleType.CTREPCM veux dire le type de valve utiliser dans cette exemple c'est un CTRE(un compagnie de module électrique pour la robotique) est c'est branché dans le PCM

Constants.brakeouvre est l'ID de la valve qui envoie l'air donc qui ouvre le piston créé dans un autre fichier appeler Constants

Constants.brakeferme est l'ID de la valve qui enlève l'air donc qui ferme le piston

```
final Solenoid pistondroite= new Solenoid(PneumaticsModuleType.REVPH, Constants.pistondroite);
```

final a déjà été définie dans d'autre documentation

Solenoid est pour dire le piston

pistondroite est le nom donner au piston

new Solenoid est pour dire que tu veux en créé un nouveau

PneumaticsModuleType.REVPH veux dire le type de valve utiliser dans cette exemple c'est un REV branché dans le PH

Constants.pistondroite est l'ID de la valve

## pour créé une fonction

pour un simple

pour un double