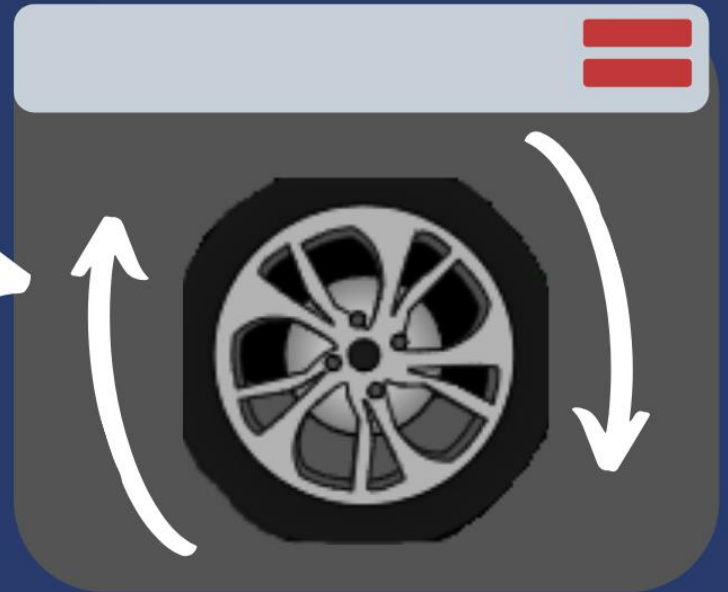
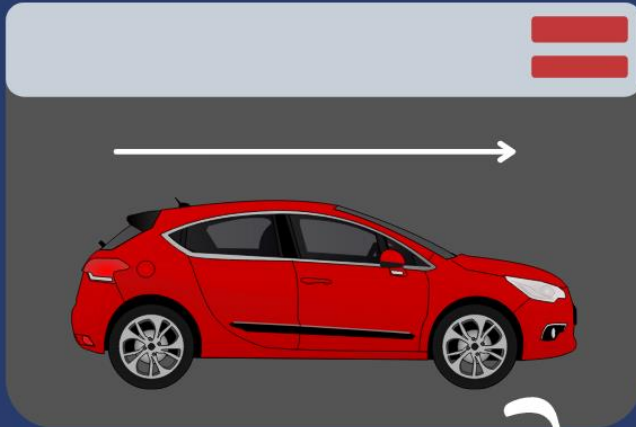


Projet N°15



# Atelier d'animation 3D

## Manuel de maintenance

DUT Informatique 2021/2022 - S4



Andréas COTTET  
Julien DERAMAIX  
Quentin TOURNIER  
Nathan VACHER



Tuteur & Commanditaire : Marc Dalmau



I. Organisation des dossiers/fichiers	5
Dossier global de l'atelier	5
Dossier du framework PyFlow	6
Dossier du packages du framework PyFlow	7
Notre package AnimationFreeCAD	8
II. Fonctionnement	9
III. Fichiers Importants	10
Classe Animation	10
Méthodes :	10
Classe Mouvement	11
Méthodes :	11
Classe NodeAnimation	13
Méthodes :	13
Classe RotationNode	14
Méthodes :	15
Classe Rotation	18
Méthodes :	19
Classe VectorPin	20
Méthodes :	21
Classe VectorInputWidget (Factories)	22
Méthodes :	23
Exemple d'un tool (AjouterEtape)	24
Code :	24
Méthodes :	24
Rendu :	24
Fichier Init.Py	25
Méthodes :	27
Outil de débogage d'un mouvement :	28
Code :	28
Rendu :	29

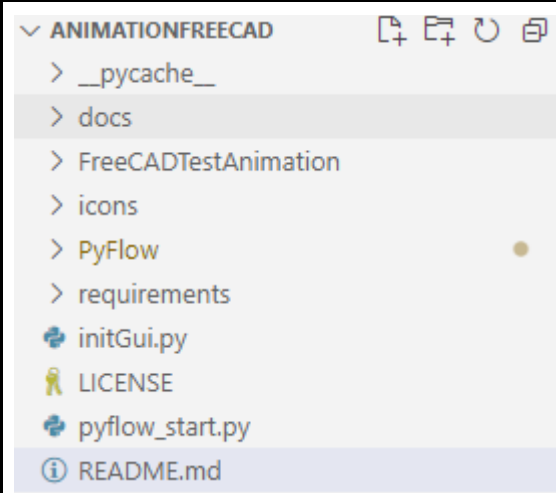


# I. Organisation des dossiers/fichiers

Notre Atelier d'Animation 3D proposé sur FreeCAD s'appuie sur un Framework de script visuel générale sur Python : PyFlow. Le dossier de notre atelier s'organise de la façon suivante.

Nous avons donc dû tenir compte des contraintes du framework ainsi que de FreeCAD pour créer notre atelier.

## Dossier global de l'atelier

	<b>Docs</b>
	<ul style="list-style-type: none"> <li>- Manuel d'installation</li> <li>- Manuel d'utilisation</li> <li>- Manuel de maintenance</li> </ul>
	<b>FreeCADTestAnimation</b>
	<ul style="list-style-type: none"> <li>- Fichier 3D et scénario du système solaire (Fichiers de test)</li> </ul>
	<b>Icons</b>
	<ul style="list-style-type: none"> <li>- L'ensemble des icônes de nos boutons</li> </ul>
	<b>PyFlow</b>
	<ul style="list-style-type: none"> <li>- Dossier du framework (Dossier principal de l'atelier)</li> </ul>
	<b>Requirements</b>
	<ul style="list-style-type: none"> <li>- L'ensemble des librairies nécessaires à l'exécution du framework</li> </ul>

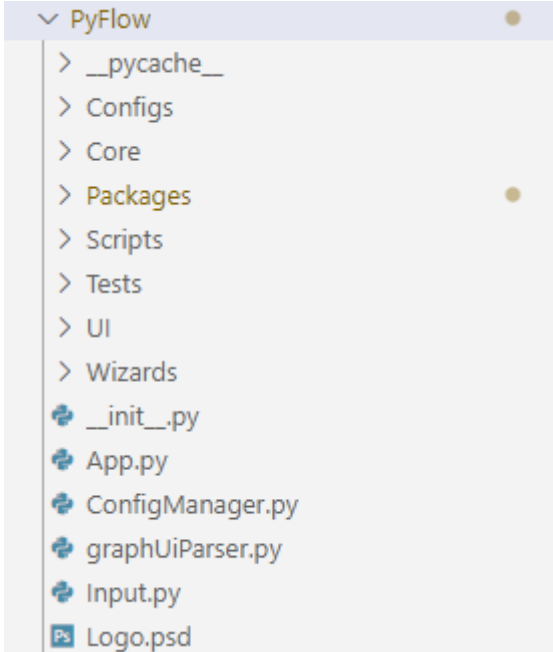
Le fichier **initGui.py** permet de créer un bouton pour lancer notre module une fois que nous avons sélectionné notre atelier dans FreeCAD.

Le fichier **LICENSE** correspond à la licence que nous utilisons : Apache Licence Version 2.0

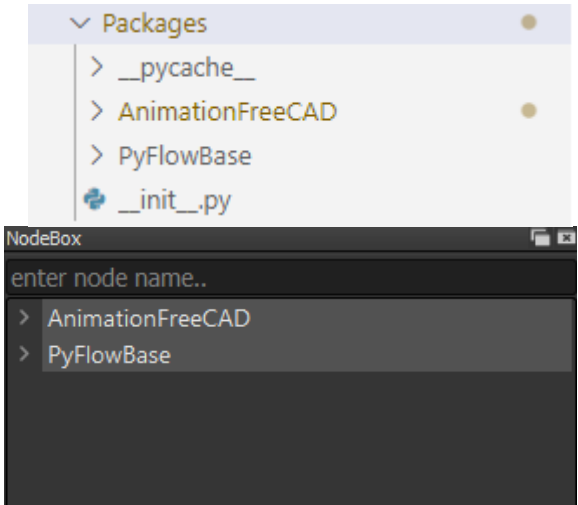
Le **pyflow\_part.py** nous permet de lancer notre module lorsqu'on appuie sur FreeCAD après la sélection de notre atelier dans FreeCAD.

Le fichier **README.md** permet d'afficher une présentation de notre module dans notre repository sur github.

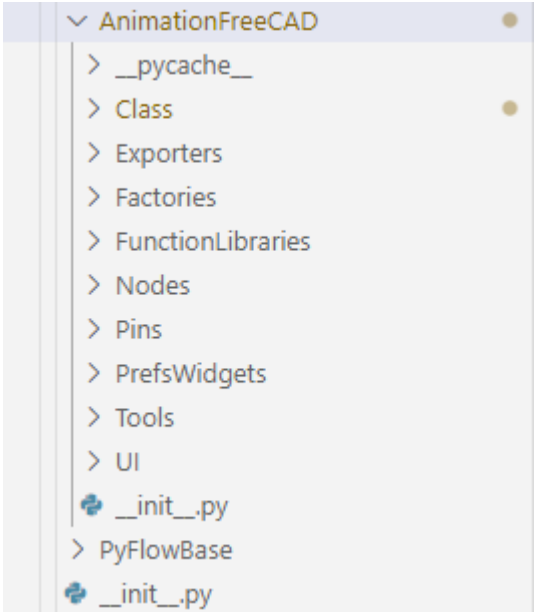
## Dossier du framework PyFlow

	<b>Configs</b>
	- Fichiers des paramètres de préférences du framework
	<b>Packages</b>
	- Dossiers où seront stockés les codes des Nodes, pins, outils, boutons, et nos algorithmes.
	<b>Test</b>
	- Fichier de test du framework
	<b>UI</b>
	- Permet de gérer toute l'interface visuelle du framework
	<b>Fichier App</b>
	- Fichier qui gère les événements basiques de l'interface - Se lance à l'ouverture de PyFlow.
	<b>Tous les autres dossiers/fichiers</b>
	- Ce sont des fichiers/dossiers générés par défaut par le Framework.

## Dossier du packages du framework PyFlow

 <p>The screenshot shows the 'Packages' panel in PyFlow. Under the 'Packages' header, there is a list: &gt; __pycache__, &gt; AnimationFreeCAD (highlighted in orange), &gt; PyFlowBase, and a file icon next to __init__.py. Below this, the 'NodeBox' window is visible, showing a text input 'enter node name..' and a list with &gt; AnimationFreeCAD and &gt; PyFlowBase.</p>	<p>PyFlow nous offre la possibilité de créer un Package qui nous permet de personnaliser l'interface et d'y implémenter du code en fonction de nos souhaits. Nous remarquons la présence d'un package de base (PyFlowBase) et du package que nous avons créé (AnimationFreeCAD).</p>
	<p>On retrouve ces packages dans l'interface de PyFlow dans le fenêtre "NodeBox". Il est possible de créer un package directement sur l'interface de PyFlow via le bouton "Plugins" puis "Create package...".</p>

## Notre package AnimationFreeCAD



Class
Factories
- Contient le code permettant de fabriquer l'interface des pins (Front)
Nodes
- Contient le code de nos nodes
Pins
- Contient le code permettant de créer le fonctionnement des pins (Back)
Tools
- Contient le code de nos outils et boutons.
Fichier __init__.py (Répertoire : AnimationFreeCAD)
- Permet de retourner les objets visuels que nous avons créés (Nodes,Outils, Pins, ...)



## II.Fonctionnement

Pour avoir un code compréhensible et lisible nous avons choisi de séparer les concepts d'animation et de mouvement.

Un mouvement correspond à un type de mouvement : il peut être une rotation, une translation, etc. Il est caractérisé par ses informations essentielles.

Exemple : Pour une translation, le mouvement est caractérisé par une courbe et le Node ou un point de départ, un point d'arrivée et un Node.

Nous sommes obligés de faire passer le Node car nous en aurons besoin en cas d'erreur pour retourner son nom.

```
self.mouvement = TranslationAvecCourbe(courbe, self)
```

Une animation correspond quant à elle aux différents modes d'affichage du mouvement. Le mouvement est caractérisé par 2 booléens : "estBoucle" si le mouvement doit être répété à l'infini et "estAller-Retour" si le mouvement doit faire un aller puis un retour et un Node.

```
self.animation = Animation(estBoucle, estAllerRetour, self)
```

Nous sommes obligés de faire passer le Node car nous en aurons besoin lorsque l'animation sera finie et devra appeler le prochaine Node.

Enfin on lance l'animation avec 2 modes, soit l'animation aura une durée, soit elle aura une vitesse.

```
self.animation.executionDuree(self.mouvement, objet, duree)
```

```
self.animation.executionVitesse(self.mouvement, objet, vitesse)
```

Dans le cas de la durée si l'animation est bouclée, elle correspond au temps que doit durer une itération de l'animation. Dans ce cas-là, l'utilisateur ne peut pas accéder aux Nodes suivants.

### III. Fichiers Importants

#### Classe Animation

```
from PyFlow.Packages.AnimationFreeCAD.Class.MouvementEnCours import MouvementEnCours
```

```
class Animation():
    def __init__(self, estBoucle, estAllerRetour, node):
        self.estBoucle = estBoucle
        self.estAllerRetour = estAllerRetour
        self.sortieNode = node.sortieNode

    def executionDuree(self, unMouvement, unObjet, uneDuree):
        unMouvement.calculTrajectoire(self.estAllerRetour, uneDuree)
        self.execution(unMouvement, unObjet)

    def executionVitesse(self, unMouvement, unObjet, uneVitesse):
        duree = unMouvement.calculDuree(uneVitesse)
        unMouvement.calculTrajectoire(self.estAllerRetour, duree)
        self.execution(unMouvement, unObjet)

    def execution(self, unMouvement, unObjet):
        unMouvement.setObjet(unObjet)
        if(self.estBoucle and self.estAllerRetour):
            unMouvement.executionBoucleAllerRetour()
        elif(self.estBoucle and not self.estAllerRetour):
            unMouvement.executionAllerBoucle()
        elif(not self.estBoucle and self.estAllerRetour):
            unMouvement.executionAllerRetour()
        else:
            unMouvement.executionAller()
```

Méthodes :

**executionDuree** → Permet de lancer une animation avec une durée.

**executionVitesse** → Permet de lancer une animation avec une vitesse.

**execution** → Permet de lancer une animation selon son mode d'animation ( Aller-Retour , Boucle ) une fois que sa vitesse ou sa durée a été calculée.

## Classe Mouvement

```

from abc import ABC
from Qt.QtWidgets import *   Import "Qt.QtWidgets" could not be resolved

import time

RAFRAICHISSEMENT = 20
NOMBRE_D_OR = 32
DEFAULT_VALUE_OBJECT_PIN = "---Select object---"

class Mouvement(ABC):
    def __init__(self, unNode):
        self.timer = None
        self.etape = 0
        self.objet = None

        self.sortieNode = unNode.sortieNode

    def mettreEnPause(self):
        if(self.timer.isActive()):
            self.timer.stop()

    def activerMouvement(self):
        if(not self.timer.isActive()):
            self.timer.start()

    def getEtape(self):
        return self.etape

    def getEtapeMax(self):
        return self.nbrPoints

    def setObjet(self, objet):
        self.objet = objet

    def executionAller(self, sortie=""):
        self.execution(True,sortie)
        self.monTemps = time.time()

    def executionAllerRetour(self, sortie=""):
        self.execution(True,"self.execution(False, \"\""+ sortie + "\"")
        self.monTemps = time.time()

    def executionAllerBoucle(self):
        self.execution(True, "self.executionAllerBoucle()")
        self.monTemps = time.time()

    def executionBoucleAllerRetour(self):
        self.executionAllerRetour("self.executionBoucleAllerRetour()")
        self.monTemps = time.time()

    def memeMouvement(self,unMouvement):
        resultat = False
        if(self.objet == unMouvement.objet):
            resultat = True
        return resultat

```

### Méthodes :

**mettreEnPause** → Permet de mettre en pause le mouvement.

**activerMouvement** → Permet de reprendre l'animation après une mise en pause.

**GetEtape** → Renvoie l'étape courante d'un mouvement.

**GetEtapeMax** → Renvoie l'étape maximale d'un mouvement.

**ExecutionAller** → Permet l'exécution d'une animation en prenant en compte simplement l'aller

**ExecutionAllerRetour** → Permet l'exécution d'une animation en prenant en compte le mouvement pour l'aller et le retour

**ExecutionAllerBoucle** → Permet l'exécution d'une animation en prenant en compte simplement l'aller mais cette fois qui se répétera indéfiniment

**ExecutionBoucleAllerRetour** → Permet l'exécution d'une animation en prenant en compte le mouvement pour l'aller et le retour mais cette fois qui se répétera indéfiniment

**MemeMouvement** → Permet de vérifier si un objet est déjà en mouvement.

## Classe NodeAnimation

La classe “NodeAnimation”, nous permet de centraliser du code dans une classe mère afin d’éviter de réécrire ce code dans chaque Node, car une grande partie de nos Nodes d’animation prennent les mêmes données d’entrées.

```
from PyFlow.Core import NodeBase
from PyFlow.Packages.AnimationFreeCAD.Class.Animation import Animation
from PyFlow.Packages.AnimationFreeCAD.Class.Mouvement import *

class NodeAnimation(NodeBase):
    def __init__(self, name):
        super().__init__(name)
        self.createInputPin("inExec", "ExecPin", None, self.compute)
        self.sortieNode = self.createOutputPin("outExec", "ExecPin")
        self.objet = self.createInputPin("Objet", "ObjectPin", DEFAULT_VALUE_OBJECT_PIN)
        self.createInputPin("Boucle", "BoolPin")
        self.createInputPin("Aller-retour", "BoolPin")
        self.createOutputPin("Objet use", "ObjectPin", DEFAULT_VALUE_OBJECT_PIN)
        self.createOutputPin("Position finale", "VectorPin")
        self.mouvement = None

    def compute(self):
        estBoucle = self.getData("Boucle")
        estAllerRetour = self.getData("Aller-retour")
        self.animation = Animation(estBoucle, estAllerRetour, self)
```

### Méthodes :

**Init** → Méthode qui permet d’initialiser des pins global à nos Nodes (Rotation, Translation, etc...)

Rappel :

La méthode “createInputPin” nous permet de créer simplement un point d’entrée dans le Node et de le nommer. Cette méthode prend 2 paramètres obligatoire :

- Le premier qui sera le nom que verra l’utilisateur sur le Node.
- Le second qui définit le type de la Pin.
- Le troisième paramètre permet de définir une valeur par défaut.
- Le quatrième paramètre nous permet de relier une fonction au l’exécution de la pin.

De manière analogue, il existe la méthode “createOutputPin” qui permet de créer des points de sortie dans le Node.

**Compute** → Permet de créer l’animation

**GetData** → Permet de renvoyer la valeur stockée dans un pin en passant en paramètre le nom du Pin.

## Classe RotationNode

```

from PyFlow.Packages.AnimationFreeCAD.Class.Rotation import Rotation
from PyFlow.Packages.AnimationFreeCAD.Class.Animation import Animation
from PyFlow.Packages.AnimationFreeCAD.Nodes.Fr.NodeAnimation import NodeAnimation
from PyFlow.Packages.AnimationFreeCAD.Class.Mouvement import *
from FreeCAD import Vector      Import "FreeCAD" could not be resolved
from Qt.QtWidgets import *     Import "Qt.QtWidgets" could not be resolved

import FreeCAD      Import "FreeCAD" could not be resolved

class RotationNode(NodeAnimation):
    def __init__(self, name):
        super(RotationNode, self).__init__(name)
        self.createInputPin("Axe de rotation", "VectorPin", Vector(0,0,1))
        self.createInputPin("Centre de rotation", "VectorPin")
        self.createInputPin("Angle au debut de la rotation", "FloatPin")
        self.createInputPin("Angle a la fin de la rotation", "FloatPin")
        self.createOutputPin("Angle final", "FloatPin")
        self.duree = self.createInputPin("Duree", "FloatPin")

    def compute(self, *args, **kwargs):
        if(self.getData("Objet") == DEFAULT_VALUE_OBJECT_PIN):
            return FenetreErreur("Erreur", self.name, self.objet.name, "Veuillez choisir un objet à mouvoir.")
        if(self.getData("Duree") <= 0):
            return FenetreErreur("Erreur", self.name, self.duree.name, "La durée ne peut pas être inférieure ou égale à 0.")

        objet = FreeCAD.ActiveDocument.getObjectsByLabel(self.getData("Objet"))[0]
        axeDeRotation = self.getData("Axe de rotation")
        centreDeRotation = self.getData("Centre de rotation")
        angleDeDebut = self.getData("Angle au debut de la rotation")
        angleDeFin = self.getData("Angle a la fin de la rotation")
        duree = self.getData("Duree")

        super().compute()

        self.mouvement = Rotation(axeDeRotation, centreDeRotation, angleDeDebut, angleDeFin, self)
        self.animation.executionDuree(self.mouvement, objet, duree)

        self.setData("Position finale", objet.Placement.Base)
        self.setData("Angle final", angleDeFin)
        self.setData("Objet use", objet.Label)

    @staticmethod
    def category():
        return 'fr|Rotation|Durée'

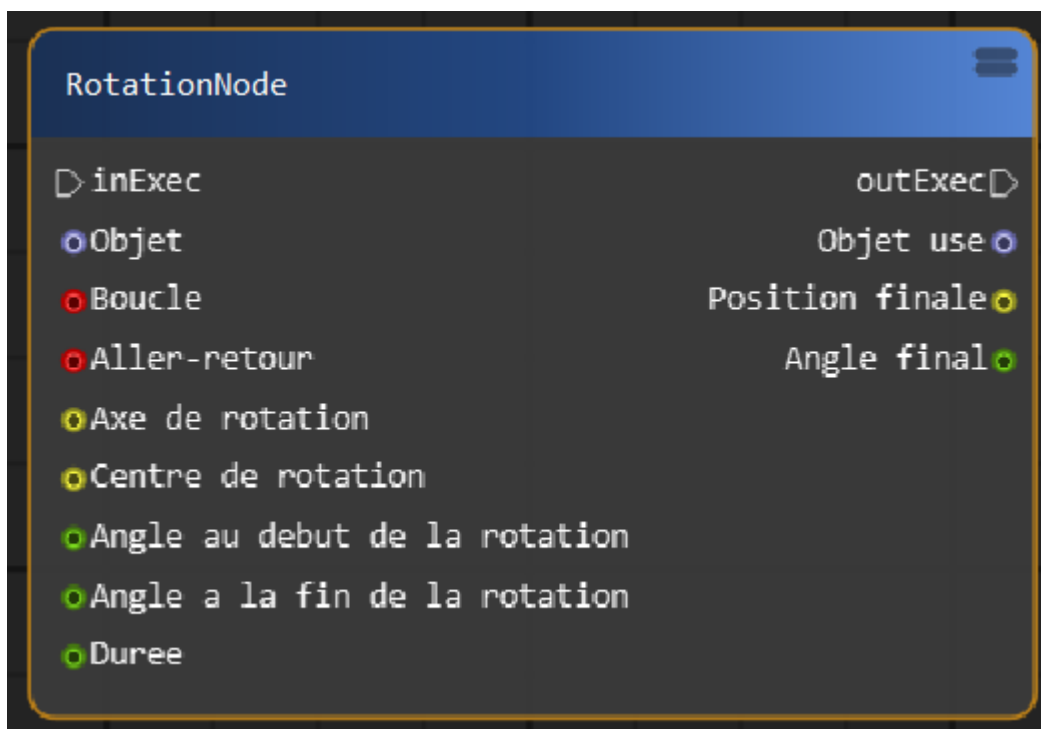
    @staticmethod
    def description():
        return "Fait tourner des objets."

```

## Méthodes :

**Init** → Permet la construction du Node "RotationNode".

```
def __init__(self, name):
    super(RotationNode, self).__init__(name)
    self.createInputPin("Axe de rotation", "VectorPin", Vector(0,0,1))
    self.createInputPin("Centre de rotation", "VectorPin")
    self.createInputPin("Angle au debut de la rotation", "FloatPin")
    self.createInputPin("Angle a la fin de la rotation", "FloatPin")
    self.createOutputPin("Angle final", "FloatPin")
    self.duree = self.createInputPin("Duree", "FloatPin")
```



**Compute** → Code exécuté lorsqu'on exécute le Node, c'est cette méthode qui le caractérise. Elle peut être découpée en 4 parties :

- Une partie de gestion des erreurs, pour vérifier les valeurs passées en paramètres

```
if(self.getData("Objet") == DEFAULT_VALUE_OBJECT_PIN):
    return FenetreErreur("Erreur", self.name, self.objet.name, "Veuillez choisir un objet à mouvoir.")
if(self.getData("Duree") <= 0):
    return FenetreErreur("Erreur", self.name, self.duree.name, "La durée ne peut pas être inférieure ou égale à 0.")
```

- Une deuxième qui nous permet de récupérer les valeurs des pins des paramètres du mouvement :

```
objet = FreeCAD.ActiveDocument.getObjectsByLabel(self.getData("Objet"))[0]
axeDeRotation = self.getData("Axe de rotation")
centreDeRotation = self.getData("Centre de rotation")
angleDeDebut = self.getData("Angle au debut de la rotation")
angleDeFin = self.getData("Angle a la fin de la rotation")
duree = self.getData("Duree")
```

- Une troisième qui récupère les paramètres de l'animation et la crée(super.compute()) le mouvement et lance l'animation :

```
super().compute()
```

```
self.mouvement = Rotation(axeDeRotation, centreDeRotation, angleDeDebut, angleDeFin,self)
self.animation.executionDuree(self.mouvement, objet, duree)
```

- La dernière partie, nous permettant de mettre les valeurs dans les pins de sortie :

```
self.setData("Position finale", objet.Placement.Base)
```

```
self.setData("Angle final", angleDeFin)
```

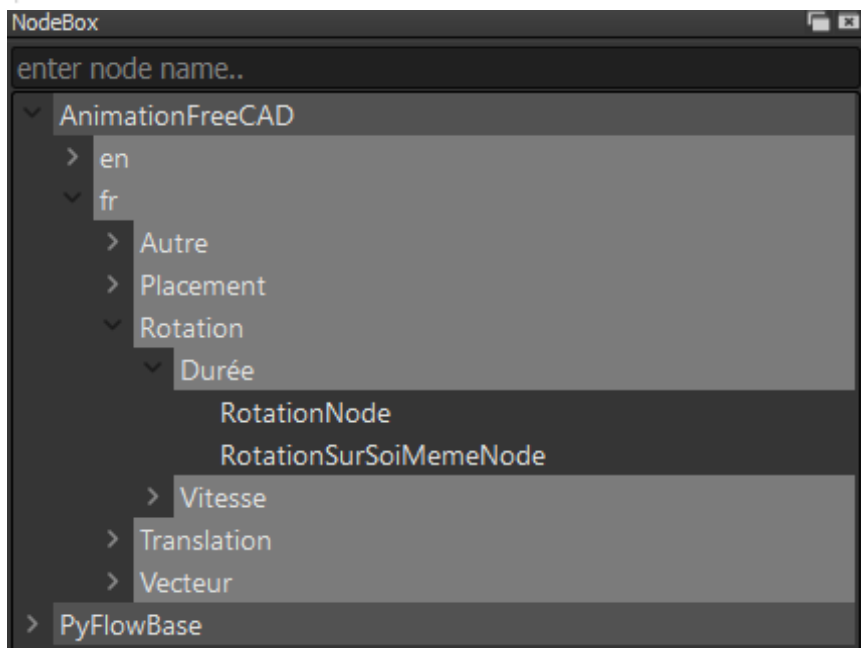
```
self.setData("Objet use", objet.Label)
```

**Category** → Permet de définir la localisation du Node dans l'arborescence de PyFlow via une chaîne de caractères :

```
@staticmethod
```

```
def category():
```

```
    return 'fr|Rotation|Durée'
```

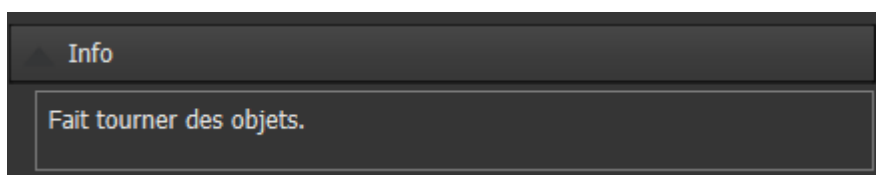


**Description** → Permet de donner une description à notre Node.

```
@staticmethod
```

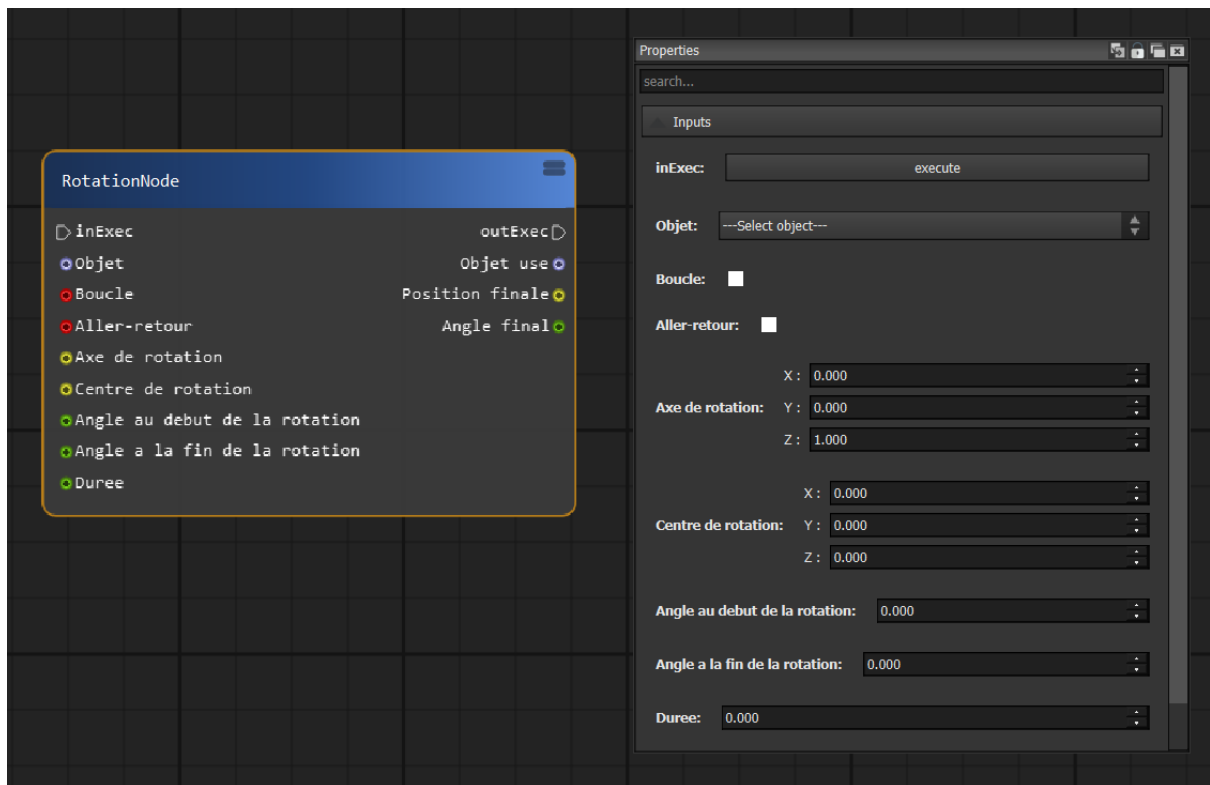
```
def description():
```

```
    return "Fait tourner des objets."
```





Rendu finale :



## Classe Rotation

```

from PyFlow.Packages.AnimationFreeCAD.Class.Mouvement import *
from PyFlow.Packages.AnimationFreeCAD.Class.MouvementEnCours import MouvementEnCours
from PySide import QtCore    Import "PySide" could not be resolved

import functools
import time
import math

class Rotation(Mouvement):

    def __init__(self, axeDeRotation, centreDeRotation, angleDeDebut, angleDeFin, unNode):
        super().__init__(unNode)
        self.axeDeRotation = axeDeRotation
        self.centreDeRotation = centreDeRotation
        self.angleDeDebut = angleDeDebut
        self.angleDeFin = angleDeFin
        self.mouvementAEteBoucle = False

    def calculTrajectoire(self, estAllerRetour, duree):
        if(estAllerRetour):
            duree = duree / 2
        self.nbrPoints = round(NOMBRE_D_OR * duree)
        self.angleAREpeter = (
            self.angleDeFin - self.angleDeDebut) / self.nbrPoints

    def calculDuree(self, uneVitesse):
        duree = (self.angleDeFin - self.angleDeDebut) / round(uneVitesse)
        return duree

    def mouvement(self, sens, suite):
        self.objet.Placement.rotate(self.centreDeRotation, self.axeDeRotation, self.angleAREpeterCourant)
        if(sens):
            self.etape += 1
            stop = self.nbrPoints
        else:
            self.etape -= 1
            stop = -1

        #print("Etape : "+ str(self.etape))

        if(self.etape == stop):
            #print(time.time() - self.monTemps)
            self.timer.stop()
            MouvementEnCours.getInstance().enleverNode(self)
            if(suite == ""):
                self.sortieNode.call()
            else:
                exec(suite)

```

```
def execution(self, sens, paramSuite, etape=-1):
    if(etape == -1):
        if(sens):
            self.etape = 0
            self.objet.Placement.Rotation.Angle = math.radians(
                self.angleDeDebut)
        else:
            self.etape = self.nbrPoints - 1
            self.objet.Placement.Rotation.Angle = math.radians(
                self.angleDeFin)

    if(sens):
        self.angleAREpeterCourant = self.angleAREpeter
    else:
        self.angleAREpeterCourant = -self.angleAREpeter

    mouvement = functools.partial(
        self.mouvement, sens=sens, suite=paramSuite)

    # Bug de timer lorsque le mouvement est un aller boucle, il se mets à avancer de plus en vite
    # Test : Lorsqu'on fait 2 aller à la suite le 2ème est accéléré, pourquoi ?
    MouvementEnCours.getInstance().ajouterNode(self)

    self.timer = QtCore.QTimer()
    self.timer.setInterval(20)

    self.timer.timeout.connect(mouvement)
    self.timer.start()

def allerALEtape(self, etape):
    deltaEtape = self.etape - etape
    if(deltaEtape >= 0):
        self.objet.Placement.rotate(
            self.centreDeRotation, self.axeDeRotation, -self.angleAREpeterCourant)
    else:
        self.objet.Placement.rotate(
            self.centreDeRotation, self.axeDeRotation, self.angleAREpeterCourant)
    self.etape = etape
```

Méthodes :

**calculTrajectoire** → Permet de calculer la trajectoire

Dans cet exemple, de combien l'angle doit être incrémenté lors de l'animation

**calculDuree** → Renvoie la durée de la rotation en fonction d'une vitesse

**mouvement** → Permet d'effectuer une étape du mouvement

**execution** → Permet de paramétrer le mouvement à répéter pour qu'il convienne à ce qui est demandé par le mode d'animation.

**allerALEtape** → Permet de placer l'objet dans une position spécifique, grâce à l'étape.

## Classe VectorPin

```
from PyFlow.Core import PinBase
from PyFlow.Core.Common import *
from FreeCAD import Vector      Import "FreeCAD" could not be resolved

import json

class VectorEncoder(json.JSONEncoder):
    def default(self, vecteur):
        if isinstance(vecteur, Vector):
            return {Vector.__name__: [vecteur.x, vecteur.y, vecteur.z]}
        json.JSONEncoder.default(self, vecteur)

class VectorDecoder(json.JSONDecoder):
    def __init__(self, *args, **kwargs):
        super(VectorDecoder, self).__init__(object_hook=self.object_hook, *args, **kwargs)

    def object_hook(self, vecteurDict):
        return Vector(vecteurDict[Vector.__name__])

def setDataG(self, data):
    super(self.__class__, self).setData(data)

class VectorPin(PinBase):
    def __init__(self, name, parent, direction, **kwargs):
        super(VectorPin, self).__init__(name, parent, direction, **kwargs)

    @staticmethod
    def IsValuePin():
        return True

    @staticmethod
    def supportedDataTypes():
        return ('VectorPin',)

    @staticmethod
    def color():
        return (200, 200, 50, 255)
```

```

@staticmethod
def pinDataTypeHint():
    return 'VectorPin', Vector()

@staticmethod
def jsonEncoderClass():
    return VectorEncoder

@staticmethod
def jsonDecoderClass():
    return VectorDecoder

@staticmethod
def internalDataStructure():
    return Vector

@staticmethod
def processData(data):
    return VectorPin.internalDataStructure()(data)

def setData(self, data):
    setDataG(self, data)

```

### Méthodes :

Pour créer notre pin vecteur nous avons dû utiliser du JSON, cela explique la création de deux classes dédiées à l'encodage et le décodage.

**Class VectorEncoder** → Classe qui permet d'encoder l'objet en JSON

**Class VectorDecoder** → Classe qui permet de décoder le JSON en objet

**setDataG** → Rédéfinition de la méthode parente en fonction de la classe permettant de mettre la valeur renseignée à l'intérieur du Node.

**isValuePin** → Permet de savoir si la pin manipule des données

**supportedDataTypes** → Permet de renvoyer le type de données supporter par la pin.

**color** → Permet de définir la couleur du pin, ici de couleur jaune.

**pinDataTypeHint** → Permet de savoir les types utilisé par la pin

**jsonEncoderClass** → Permet de définir la class charger d'encoder le json.

**jsonDecoderClass** → Permet de définir la classe chargée de décoder le json.

**internalDataStructure** → Permet de renvoyer le type du pin, ici un vecteur.

**processData** → Permet de transformer les données du pin dans le bon type.

**setData** → Appelle à la fonction setDataG.

## Classe VectorInputWidget (Factories)

```
class VectorInputWidget(InputWidgetRaw):
    def __init__(self, **kws):
        super(VectorInputWidget, self).__init__(**kws)
        self.setLayout(QtWidgets.QGridLayout())
        self.x = valueBox(None, "float", True)
        self.y = valueBox(None, "float", True)
        self.z = valueBox(None, "float", True)
        self.labelX = QLabel("X :")      "QLabel" is not defined
        self.labelY = QLabel("Y :")      "QLabel" is not defined
        self.labelZ = QLabel("Z :")      "QLabel" is not defined

        self.layout().addWidget(self.labelX, 1, 1)
        self.layout().addWidget(self.x, 1, 2)
        self.layout().addWidget(self.labelY, 2, 1)
        self.layout().addWidget(self.y, 2, 2)
        self.layout().addWidget(self.labelZ, 3, 1)
        self.layout().addWidget(self.z, 3, 2)

        self.layout().setColumnStretch(2, 1)

        self.x.valueChanged.connect(self._onDataChangedX)
        self.y.valueChanged.connect(self._onDataChangedY)
        self.z.valueChanged.connect(self._onDataChangedZ)

    def blockWidgetSignals(self, blocked):
        for i in [self.x, self.y, self.z]:
            i.blockSignals(blocked)

    def asDataTypeClass(self):
        return FreeCAD.Vector([self.x.value(), self.y.value(), self.z.value()])

    def _onDataChangedX(self, val):
        tmp = self.asDataTypeClass()
        tmp.x = val
        self.dataSetCallback(tmp)

    def _onDataChangedY(self, val):
        tmp = self.asDataTypeClass()
        tmp.y = val
        self.dataSetCallback(tmp)

    def _onDataChangedZ(self, val):
        tmp = self.asDataTypeClass()
        tmp.z = val
        self.dataSetCallback(tmp)

    def setWidgetValue(self, val):
        self.x.setValue(val.x)
        self.y.setValue(val.y)
        self.z.setValue(val.z)
```

## Méthodes :

On y retrouve un constructeur dans lequel on fabrique l'interface des pins grâce à la classe VecteurPin que l'on retrouve dans le dossier "Pins"

**blockWidgetSignals** → Permet de bloquer les input si jamais des valeurs sont branchées dessus

**init** → Constructeur qui nous permet de créer la partie visuel des pins qui sera dans la fenêtre propriété.

**asDataTypeClass** → Permet de retourner le bon type de donnée en sortie de la pin

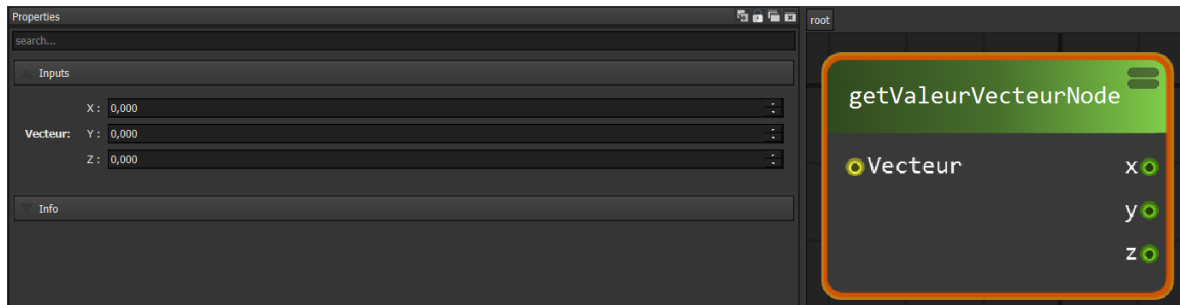
**\_onDataChangedX** → Permet de définir la valeur de x quand x est changé

**\_onDataChangedY** → Permet de définir la valeur de y quand y est changé

**\_onDataChangedZ** → Permet de définir la valeur de z quand z est changé

**setWidgetValue** → Permet de mettre à jour les valeurs des composantes de la pin

Voici un exemple de Node qui utilise la classe vecteurPin.



## Exemple d'un tool (AjouterEtape)

Code :

```
from PyFlow.Packages.AnimationFreeCAD.Class.Coordonnees import Coordonnees
from nine import str
from PyFlow.UI.Tool.Tool import ShelfTool
from PyFlow.Packages.AnimationFreeCAD.Class.MouvementEnCours import MouvementEnCours
from Qt import QtGui      Import "Qt" could not be resolved

import os
scriptDir = os.path.dirname(os.path.realpath(__file__))

class AjouterEtape(ShelfTool):
    """docstring for ResetPosition."""
    def __init__(self):
        super(AjouterEtape, self).__init__()

    @staticmethod
    def toolTip():
        return "Ajoute une étape à la liste avec les paramètres de tous les objets de la fenêtre FreeCAD"

    @staticmethod
    def getIcon():
        path = "../../../icons/boutonAjouterEtape.svg"
        return QtGui.QIcon(scriptDir + os.path.sep + path)

    @staticmethod
    def name():
        return str("AjouterEtape")

    def do(self):
        print("Est ce que ça marche ?")
        Coordonnees.getInstance().ajouterEtape()
```

Méthodes :

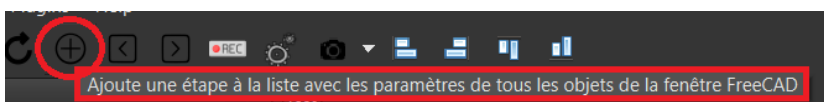
**ToolTip** → Permet de renvoyer la description du bouton au survole

**GetIcon** → Permet de renvoyer l'image bouton

**Name** → Permet de renvoyer le nom du bouton

**Do** → Permet de définir l'action associé au bouton lors du clic

Rendu :





## Fichier Init.Py

Chaque dossier du répertoire Packages contient un fichier appelé init.py. Son but est de retourner les objets visuels que nous pouvons utiliser dans notre module, c'est-à-dire, les Nodes, les pins, les boutons, ... que nous avons créés. Dans ce fichier nous devons donc importer les éléments graphiques que nous avons créés et les intégrer dans leurs listes correspondantes.

```
PACKAGE_NAME = 'AnimationFreeCAD'

from collections import OrderedDict
from PyFlow.UI.UIInterfaces import IPackage

# Pins
from PyFlow.Packages.AnimationFreeCAD.Pins.VectorPin import VectorPin
from PyFlow.Packages.AnimationFreeCAD.Pins.ObjectPin import ObjectPin
from PyFlow.Packages.AnimationFreeCAD.Pins.CurvePin import CurvePin

# Function based nodes

# Class based nodes
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.TranslationRectiligneNode import TranslationRectiligneNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.TranslationRectiligneVitesseNode import TranslationRectiligneVitesseNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.TranslationAvecCourbeNode import TranslationAvecCourbeNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.TranslationAvecCourbeVitesseNode import TranslationAvecCourbeVitesseNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.RotationSurSoiMemeNode import RotationSurSoiMemeNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.RotationSurSoiMemeVitesseNode import RotationSurSoiMemeVitesseNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.RotationNode import RotationNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.RotationVitesseNode import RotationVitesseNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.getValeurVecteurNode import getValeurVecteurNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.PlacerNode import PlacerNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.setAngleObjectNode import setAngleObjectNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.CommencerNode import CommencerNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.FonctionMathNode import FonctionMathNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.TranslationAccelere import TranslationAccelere
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.TranslationDecelere import TranslationDecelere
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.Attendre import Attendre
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.CreateVecteurNode import CreateVecteurNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.MakeVecteurNode import MakeVecteurNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.RotationXNode import RotationXNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.RotationYNode import RotationYNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.fr.RotationZNode import RotationZNode

from PyFlow.Packages.AnimationFreeCAD.Nodes.en.RectilinearTranslationNode import RectilinearTranslationNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.RectilinearTranslationBySpeedNode import RectilinearTranslationBySpeedNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.TranslationWithCurveNode import TranslationWithCurveNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.TranslationWithCurveBySpeedNode import TranslationWithCurveBySpeedNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.SpinAroundNode import SpinAroundNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.SpinAroundBySpeedNode import SpinAroundBySpeedNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.SpinNode import SpinNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.SpinBySpeedNode import SpinBySpeedNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.getVectorValueNode import getVectorValue
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.PlaceNode import PlaceNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.setAngleObjectNode import setAngleObjectNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.StartNode import StartNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.AcceleratesTranslationNode import AcceleratesTranslationNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.DeceleratesTranslationNode import DeceleratesTranslationNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.WaitNode import WaitNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.CreateVectorNode import CreateVectorNode
from PyFlow.Packages.AnimationFreeCAD.Nodes.en.MakeVectorNode import MakeVectorNode

# Tools
from PyFlow.Packages.AnimationFreeCAD.Tools.MettreEnPause import MettreEnPause
from PyFlow.Packages.AnimationFreeCAD.Tools.ContinuerMouvements import ContinuerMouvements
from PyFlow.Packages.AnimationFreeCAD.Tools.ResetPosition import ResetPosition
from PyFlow.Packages.AnimationFreeCAD.Tools.AjouterEtape import AjouterEtape
from PyFlow.Packages.AnimationFreeCAD.Tools.EtapePrecedente import EtapePrecedente
```

Fichier Init.py partie importation ci-dessus

```

_NODES = {
    TranslationRectiligneNode.__name__: TranslationRectiligneNode,
    TranslationAvecCourbeNode.__name__: TranslationAvecCourbeNode,
    RotationSurSoiMemeNode.__name__: RotationSurSoiMemeNode,
    CommencerNode.__name__: CommencerNode,
    RotationNode.__name__: RotationNode,
    getValeurVecteurNode.__name__: getValeurVecteurNode,
    PlacerNode.__name__: PlacerNode,
    setAngleObjectNode.__name__: setAngleObjectNode,
    TranslationRectiligneVitesseNode.__name__: TranslationRectiligneVitesseNode,
    TranslationAvecCourbeVitesseNode.__name__: TranslationAvecCourbeVitesseNode,
    RotationSurSoiMemeVitesseNode.__name__: RotationSurSoiMemeVitesseNode,
    RotationVitesseNode.__name__: RotationVitesseNode,
    FonctionMathNode.__name__: FonctionMathNode,
    TranslationDecelere.__name__: TranslationDecelere,
    TranslationAccelere.__name__: TranslationAccelere,
    Attendre.__name__: Attendre,
    CreateVecteurNode.__name__: CreateVecteurNode,
    MakeVecteurNode.__name__: MakeVecteurNode,
    RotationXNode.__name__: RotationXNode,
    RotationYNode.__name__: RotationYNode,
    RotationZNode.__name__: RotationZNode,

    RectilinearTranslationNode.__name__: RectilinearTranslationNode,
    RectilinearTranslationBySpeedNode.__name__: RectilinearTranslationBySpeedNode,
    TranslationWithCurveNode.__name__: TranslationWithCurveNode,
    TranslationWithCurveBySpeedNode.__name__: TranslationWithCurveBySpeedNode,
    SpinAroundNode.__name__: SpinAroundNode,
    SpinAroundBySpeedNode.__name__: SpinAroundBySpeedNode,
    SpinNode.__name__: SpinNode,
    SpinBySpeedNode.__name__: SpinBySpeedNode,
    getVectorValue.__name__: getVectorValue,
    PlaceNode.__name__: PlaceNode,
    setAngleObjectNode.__name__: setAngleObjectNode,
    StartNode.__name__: StartNode,
    AcceleratesTranslationNode.__name__: AcceleratesTranslationNode,
    DeceleratesTranslationNode.__name__: DeceleratesTranslationNode,
    WaitNode.__name__: WaitNode,
    CreateVectorNode.__name__: CreateVectorNode,
    MakeVectorNode.__name__: MakeVectorNode
}

_PINS = {
    VectorPin.__name__: VectorPin,
    ObjectPin.__name__: ObjectPin,
    CurvePin.__name__: CurvePin
}

_TOOLS = OrderedDict()
_PREFS_WIDGETS = OrderedDict()
_EXPORTERS = OrderedDict()

_TOOLS[MettreEnPause.__name__] = MettreEnPause
_TOOLS[ContinuerMouvements.__name__] = ContinuerMouvements
_TOOLS[ArreterTousLesMouvements.__name__] = ArreterTousLesMouvements
_TOOLS[ResetPosition.__name__] = ResetPosition
_TOOLS[AjouterEtape.__name__] = AjouterEtape
_TOOLS[EtapePrecedente.__name__] = EtapePrecedente
_TOOLS[EtapeSuivante.__name__] = EtapeSuivante
_TOOLS[ExportationVideo.__name__] = ExportationVideo

```

Pour qu'un Node soit pris en compte dans la fenêtre NodeBox, il faudra lui associer un nom et une classe. De même pour les boutons et les pins.

```
class AnimationFreeCAD(IPackage):
    def __init__(self):
        | super(AnimationFreeCAD, self).__init__()

    @staticmethod
    def GetExporters():
        | return _EXPORTERS

    @staticmethod
    def GetFunctionLibraries():
        | return _FOO_LIBS

    @staticmethod
    def GetNodeClasses():
        | return _NODES

    @staticmethod
    def GetPinClasses():
        | return _PINS

    @staticmethod
    def GetToolClasses():
        | return _TOOLS

    @staticmethod
    def PinsInputWidgetFactory():
        | return getInputWidget

    @staticmethod
    def PrefsWidgets():
        | return _PREFS_WIDGETS
```

Méthodes :

**GetExporters** → Permet de retourner tous les exporters

**GetFunctionLibraries** → Permet de retourner toutes les "FunctionLibrairies"

**GetNodeClasses** → Permet de retourner tous les Nodes

**GetPinClasses** → Permet de retourner toutes les classes des pins créer au préalable (Par exemple vecteurPin)

**GetToolClasses** → Permet de retourner tous les Tools(utilisé pour créer notre propre bouton)

**PinsInputWidgetFactory** → Permet de retourner les entrées nécessaires au pin

**PrefsWidgets** → Permet de retourner les préférences des widgets

## Outil de débogage d'un mouvement :

Code :

```

if(self.getMouvement() != False):
    Etape = CollapsibleFormWidget(headName="Etape")
    self.etape = QSpinBox() "QSpinBox" is not defined
    self.etape.setRange(1,self.getEtapeMax())
    self.etape.setValue(self.getEtape())
    self.avancement = QSpinBox() "QSpinBox" is not defined

    Etape.addWidget("Nbr d'étapes : "+ str(self.getEtapeMax()) + " Etape ", self.etape)

    self.etape.valueChanged.connect(lambda :self.allerALEtape(self.etape.value()))
    self.avancement.valueChanged.connect(lambda :self.etape.setSingleStep(self.avancement.value()))

    Etape.addWidget("Avancement ", self.avancement)
    self.avancement.setMinimum(1)

    self.buttonLancerMouvement = QPushButton() "QPushButton" is not defined
    self.buttonLancerMouvement.setText("Lancer mouvement")
    self.buttonLancerMouvement.clicked.connect(lambda :self.lancerMouvement(self.etape.value()))

    Etape.addWidget("Fonctions", self.buttonLancerMouvement)

    propertiesWidget.addWidget(Etape)

def allerALEtape(self, etape):
    print("test")
    mouvement = self.getMouvement()
    if(mouvement != False):
        mouvement.allerALEtape(etape - 1)
    else:
        print("Impossible")

def lancerMouvement(self, etape):
    mouvement = self.getMouvement()
    if(etape != self.getEtapeMax()):
        if(mouvement != False):
            mouvement.execution(True, "", etape - 1)

def getEtape(self):
    mouvement = self.getMouvement()
    print(mouvement)
    if(mouvement != False):
        print(mouvement.getEtape())
        return mouvement.getEtape()
    else:
        print("Pas de mouvement")
        return(-1)

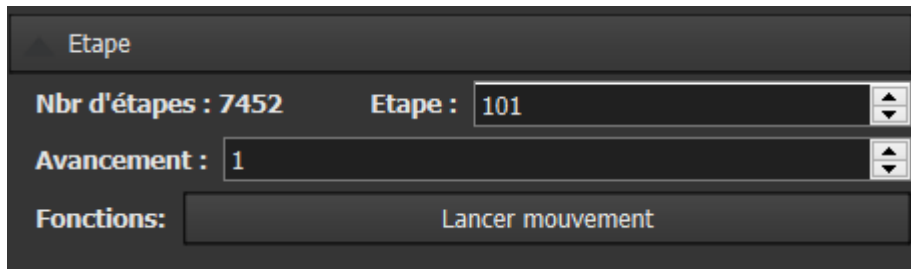
def getEtapeMax(self):
    mouvement = self.getMouvement()
    if(mouvement != False):
        return mouvement.getEtapeMax()
    else:
        print("Pas de mouvement")
        return(-1)

def getMouvement(self):
    try:
        self._rawNode.mouvement
        if self._rawNode.mouvement is not None:
            return self._rawNode.mouvement
        else:
            return False
    except AttributeError:
        return False

```

Il faut également que les méthodes getEtape, getEtapeMax et allerALEtape soient définies dans la classe du mouvement.

Rendu :



The screenshot shows a dark-themed software interface with the following elements:

- A header bar with a small upward-pointing triangle and the text "Etape".
- A row with "Nbr d'étapes : 7452" on the left and "Etape : 101" on the right, where "101" is in a text box with up/down arrow buttons.
- A row with "Avancement : 1" in a text box with up/down arrow buttons.
- A row with "Fonctions:" on the left and a button labeled "Lancer mouvement" on the right.

A retrouver dans le bloc propriétés lorsqu'un mouvement à été exécuté au moins une fois.