

# Introduction à la programmation avec

# Ropy



# Présentation de Ropy et de son environnement de programmation



**Ropy** est un rover martien qui se commande grâce au langage **Python**. L'interface de programmation se décompose en 2 fenêtres : un éditeur de texte et le simulateur.

Le **simulateur** permet de **visualiser l'évolution du Rover**.

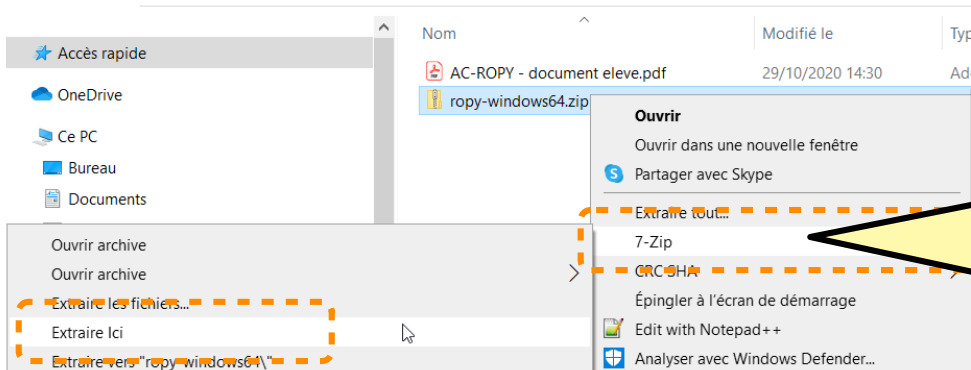


Un **éditeur de texte** (Notepad++, Spyder, Jupyter, Emacs, ...) pour **écrire le programme** en **Python**.

```
File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/phroy/Bureau/ropy-windows64/rp_cmd.py
rp_cmd.py x
1 import bge # Bibliothèque Blender Game Engine (UPBGE)
2 from rp_lib import * # Bibliothèque Ropy
3
4 #####
5 # rp_cmd.py
6 # @title: Commandes pour le Rover Ropy
7 # @project: Ropy (Blender-EduTech)
8 #####
9
10 #####
11 # Initialisation du niveau :
12 # Niveau 1 : Les premiers pas de Ropy
13 # Niveau 2 : Sécuriser Ropy
14 # Niveau 3 : Partir au bout du monde
15 # Niveau 4 : Faire face à l'inconnu
16 # Niveau 5 : Se rendre utile
17 #####
18
19 #####
20 # Fonctions
21 #####
22
23
```

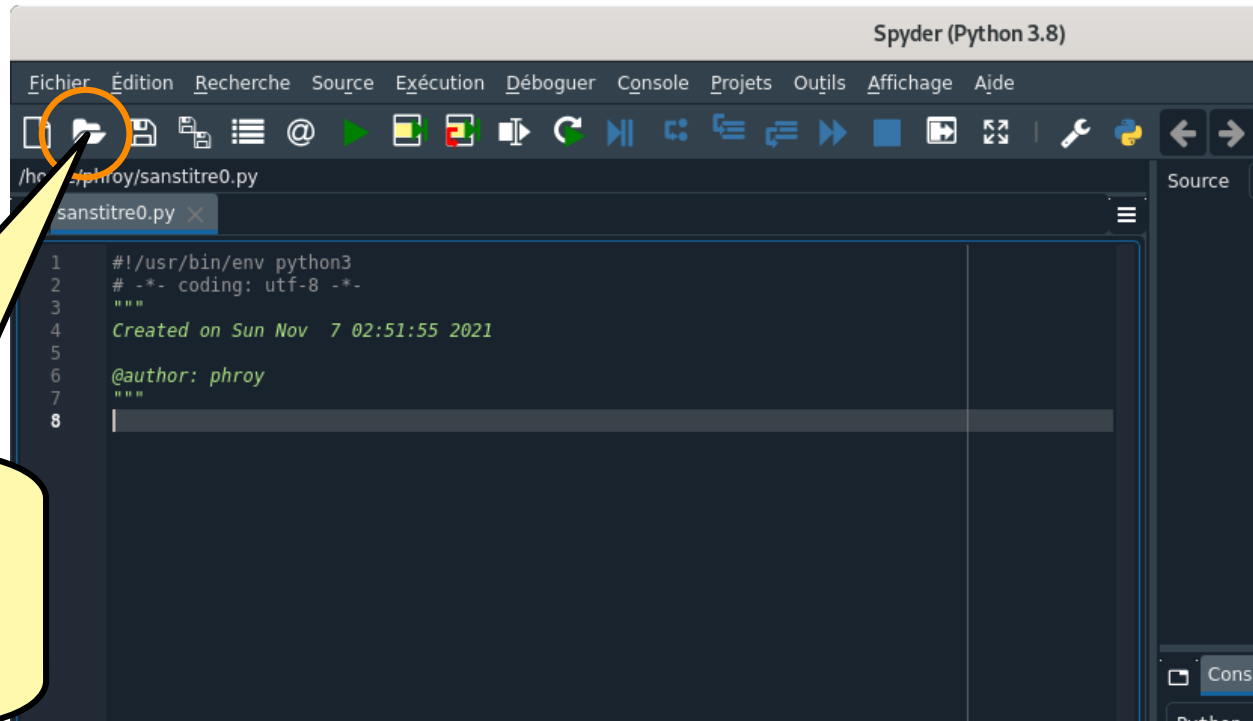
# Éditer le programme avec Spyder

## Ouvrir le fichier ropy\_cmd.py



1 : Récupérer l'archive **Ropy-2-windows64.zip** et la décompresser avec **7-Zip** dans votre répertoire.

2 : Lancer le Logiciel **Spyder**.



3: Ouvrir le fichier Python à éditer **rp\_cmd.py** (Ropy commandes).

# Éditer le programme avec Spyder

## Exécution du programme



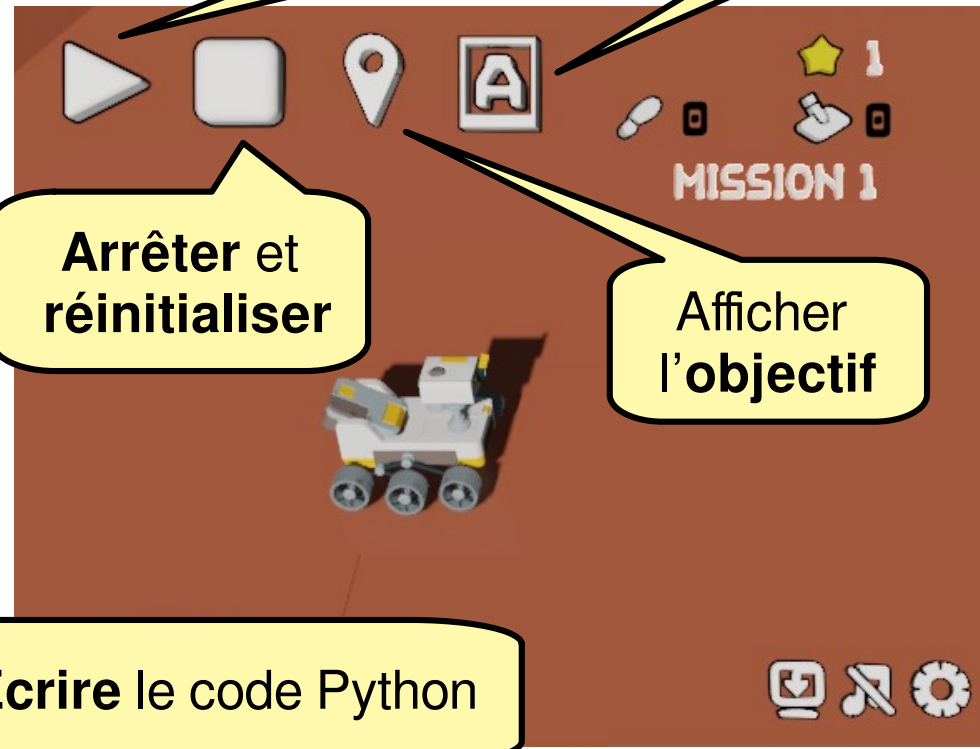
```
File Edit Search Source
/home/phroy/Bureau/ropy-win
rp_cmd.py x
1 import bge # Bibli
2 from rp_lib import
3
4 #####
5 # rp_cmd.py
6 # @title: Commandes pour le Rover Ropy
7 # @project: Ropy (Blender-EduTech)
8 #####
9
10 #####
11 # Initialisation du niveau :
12 # Niveau 1 : Les premiers pas de Ropy
13 # Niveau 2 : Sécuriser Ropy
14 # Niveau 3 : Partir au bout du monde
15 # Niveau 4 : Faire face à l'inconnu
16 # Niveau 5 : Se rendre utile
17 #####
18
19 #####
20 # Fonctions
21 #####
22
23 #####
24
25 #####
26 # Commandes
27 #####
28
29 def commandes():
30
31     rp_niveau (1) # Saisir le niveau (de 0 à 5)
32
33     rp_gauche()
34     rp_avancer()
35     rp_avancer()
36     rp_avancer()
37     rp_avancer()
38
39     rp_fin()
40
```

5 : **Sauvegarder** le fichier  
**Attention !**  
Toujours sauvegarder le fichier avant son exécution avec le simulateur.

Le simulateur est le programme **ropy.exe**

6 : **Exécuter** le programme

Afficher l'**aide**



**Arrêter et réinitialiser**

Afficher l'**objectif**

4 : **Écrire** le code Python

# Contenu du fichier ropy\_cmd.py



Le fichier `rp_cmd.py` comporte 4 sections.

```
import bge # Bibliothèque Blender Game Engine (UPBGE)
from rp_lib import * # Bibliothèque Ropy

#####
# rp_cmd.py
# @title: Commandes pour le Rover Ropy
# @project: Ropy (Blender-EduTech)
#####

#####
# Initialisation du niveau :
# Niveau 1 : Les premiers pas de Ropy
# Niveau 2 : Sécuriser Ropy
# Niveau 3 : Partir au bout du monde
# Niveau 4 : Faire face à l'inconnu
# Niveau 5 : Se rendre utile
#####

#####
# Fonctions
#####

#####
# Commandes
#####

def commandes():

    rp_niveau (1) # Saisir le niveau (de 1 à 5)

    rp_gauche()
    rp_avancer()
    rp_avancer()
    rp_avancer()
    rp_avancer()

    rp_fin()

#####
# En: Externals calls << DONT CHANGE THIS SECTION >>
# Fr: Appels externes << NE PAS MODIFIER CETTE SECTION >>
#####

if __name__=='start':
    thread_cmd_start(commandes)
if __name__=='stop':
    thread_cmd_stop()
```

Vous pouvez changer de niveau avec la commande `rp_niveau(numero_niveau)`

**Import des bibliothèques**  
**Ne pas modifier cette section**

**Fonctions** : section pour le codage de **vos** fonctions

**Commandes** : section pour le codage des commandes du robot

**Appels du simulateur**  
(Blender Game Engine)  
**Ne pas modifier cette section**



# Niveau 2 – Ma première fonction

## Création d'une fonction



**Objectif 2** : Aller au niveau 2, pour faciliter le codage, on va créer la fonction `mrp_avancer()` regroupant `avancer` et `marquer`.

La **définition d'une fonction** se fait de la manière suivante :

```
def fonction_1(arguments) :  
    instruction_1  
    instruction_2  
    ...  
    ← return valeurs_renvoyées
```

Cet espace est l'**indentation**, il se fait avec la touche tabulation (Tab).

**Attention !** C'est l'**indentation** qui définit le **début et la fin d'un bloc**.

L'**appel de la fonction** est simplement :  
`fonction_1(arguments)`

```
#####  
# Fonctions  
#####
```

```
#####  
# Commandes  
#####
```

# Niveau 3 – Apprendre le danger

## Structure conditionnelle (si, alors, sinon)



**Objectif 3.1** : Aller au niveau 3, provoquer une collision avec un obstacle en avançant et observer ce qu'il se passe. Il semble assez clair qu'il faut sécuriser l'avance du robot.

Si le test de `condition` est vrai  
**alors** exécuter `instruction_1`  
**sinon** exécuter `instruction_2`

```
#####  
# Commandes  
#####  
  
_____  
  
_____  
  
_____  
  
_____
```

Une **structure conditionnelle** permet d'exécuter des instructions en fonction du résultat d'un test (condition).

```
if condition :  
    instructions_1  
else :  
    instructions_2
```

le **sinon**  
n'est pas  
obligatoire

Les conditions peuvent être

- `a == b` : a est égal à b
- `a != b` : a est différent de b
- `a < b` : a est strictement inférieur à b
- `a <= b` : a est inférieur ou égal à b
- `a == b and c == d` : les deux conditions doivent être vrai (fonction ET)
- `a == b or c == d` : une des deux conditions doit être vrai (fonction OU)

La fonction pour **détecter un obstacle** est : `rp_detect()`. La fonction retourne **True** si il a un mur et **False** si il n'y a pas de mur.







# Niveau 4 – Partir au bout du monde

## Passage d'argument (dans une fonction)



**Objectif 4.2** : Afin de faciliter le code nous allons créer une fonction pour avancer d'un nombre de pas : `mrp_avancer_pas (nb_pas)`.

Lors de la définition de fonction `mrp_avancer()`, nous n'avons pas utilisé les arguments. Un **argument** est une variable qui permet de **paramétrer la fonction**.

Par exemple : une fonction pour faire tourner le robot à partir de valeur angulaire.

```
def mrp_tourner(angle):  
    if angle == 90:  
        rp_droite()  
    if angle == -90:  
        rp_gauche()  
    if angle==180 or angle==-180:  
        rp_droite()  
        rp_droite()
```

angle est  
l'argument

```
#####  
# Fonctions  
#####
```

```
#####  
# Commandes  
#####
```

# Niveau 5 - Faire face à l'inconnu

## Structure itérative - boucle indéfinie (tant que)



**Objectif 5** : Aller au niveau 5, **Ropy** doit toujours atteindre la même case, mais son lieu de départ change à chaque fois. Pour pallier à l'aléatoire, il faut créer une fonction qui permet d'atteindre un obstacle : `mrp_avancer_mur()`.

Une **boucle indéfinie** (nombre de répétitions inconnu à l'avance) se poursuit **tant qu'une condition est vraie**.

```
while condition :  
    bloc_instructions
```

Par exemple : une boucle pour activer le robot par la saisie d'un code de déverrouillage. On reste dans la boucle **tant que** la saisie n'est pas « okropy ».

```
saisie=""  
while saisie!="okropy" :  
    saisie = input()
```

`input()` permet de faire une saisie au clavier dans la console.

```
#####  
# Fonctions  
#####
```

```
#####  
# Commandes  
#####
```



