

# Séquence 5

*Comment sont définis les  
règles de fonctionnement d'un système ?*

I2D



Lycée PRAVAZ

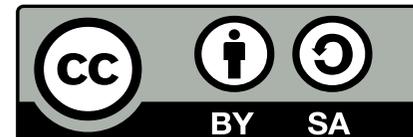
# Introduction à la programmation avec

# Ropy



**LA FORGE**

des communs  
numériques  
éducatifs



[Philippe Roy <philippe.roy@ac-grenoble.fr>](mailto:philippe.roy@ac-grenoble.fr)

# Présentation de Ropy et de son environnement de programmation

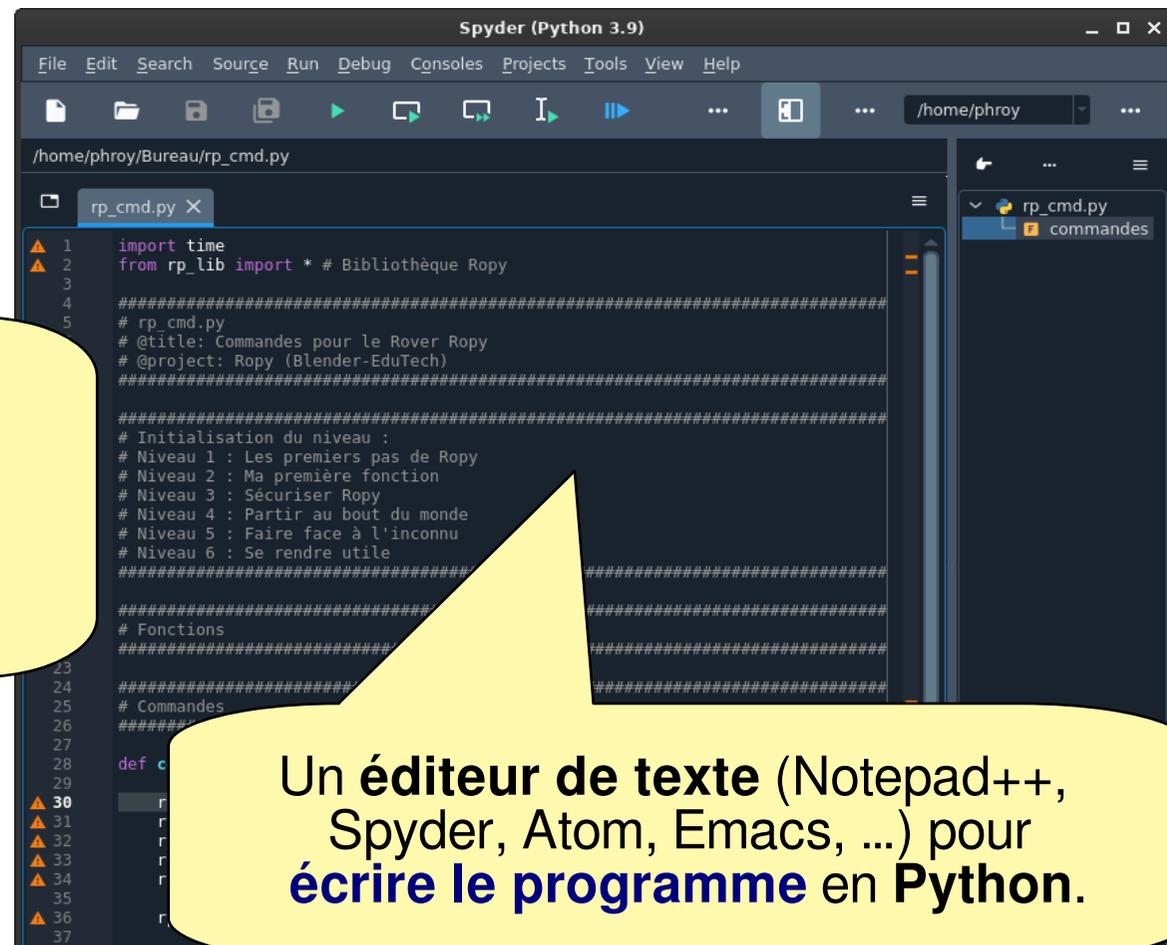
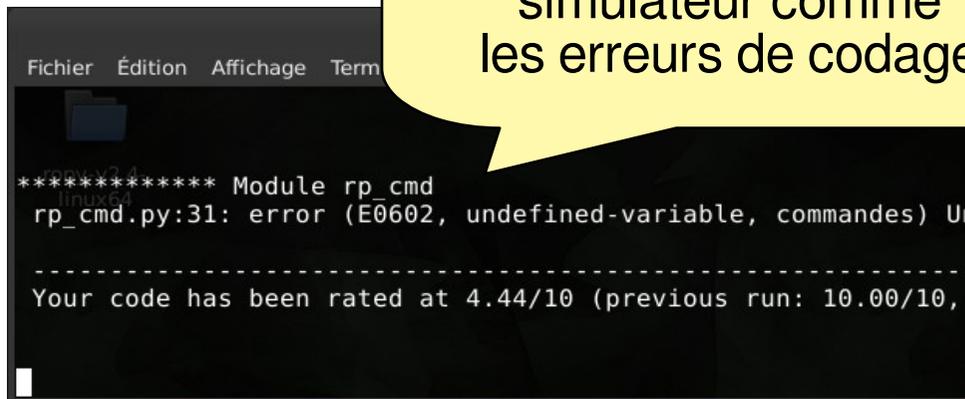


**Ropy** est un rover martien qui se commande grâce au langage **Python**. L'interface de programmation se décompose en **3 fenêtres** : un éditeur de texte, le simulateur et la console.



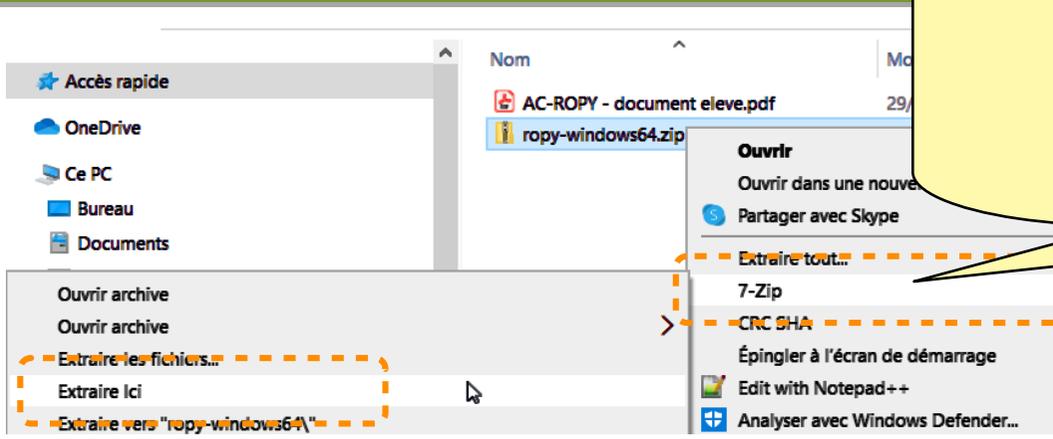
Le **simulateur** permet de **visualiser l'évolution du Rover**.

La **console** pour **visualiser les informations** du simulateur comme les erreurs de codage.



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

# Mettre en place l'environnement de développement



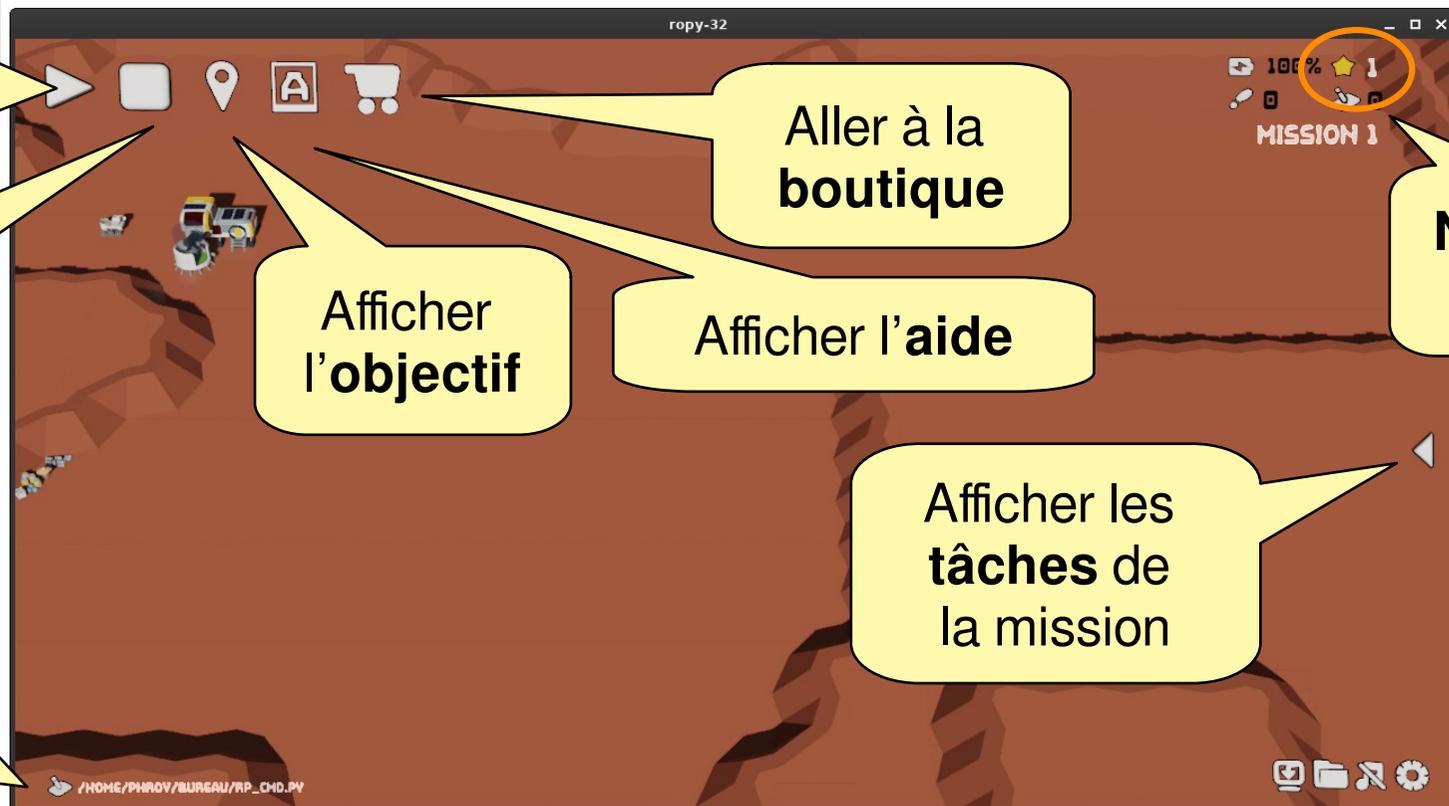
1 : Récupérer l'archive **ropy-windows64.zip** et la décompresser avec **7-Zip** sur le **bureau**. L'extraction va créer le répertoire ropy

Le **simulateur** et la **console** se lancent en même temps avec **ropy.bat** (situé dans le répertoire créé).

Exécuter le programme

Arrêter et réinitialiser

Fichier de commandes



Aller à la boutique

Afficher l'objectif

Afficher l'aide

Afficher les tâches de la mission

Niveau actuel

# Mettre en place l'environnement de développement

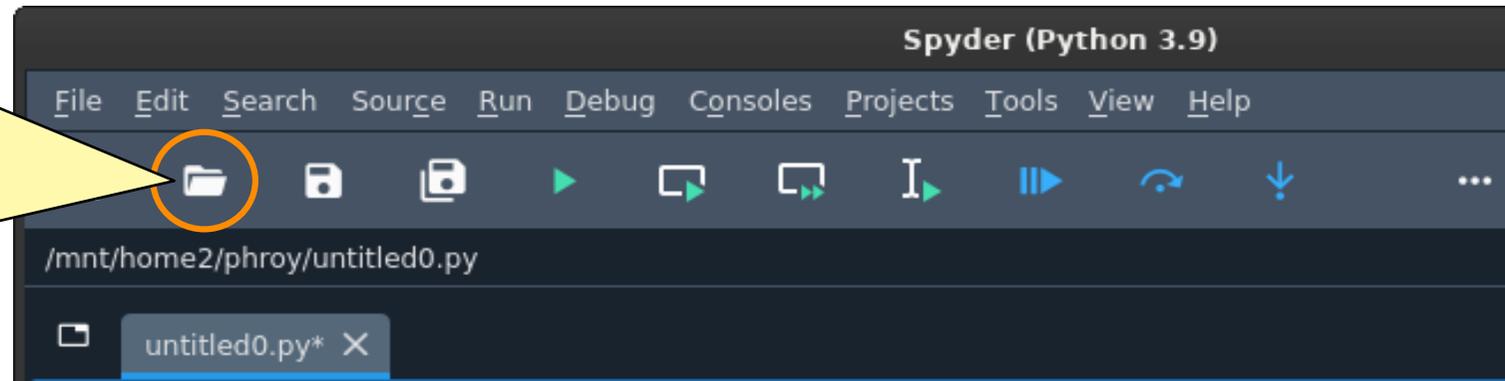


2 : Copier **dans votre répertoire** le fichier de commandes : **ropy\_cmd.py** (ropy commandes).

3 : Lancer **Spyder**.



4 : Dans **Spyder** ouvrir le fichier de commandes qui a été précédemment copié dans votre répertoire.



6 : Le nom de votre fichier doit apparaître ici.

5 : Dans le **simulateur**, définir votre fichier comme fichier de commandes.



# Mettre en place l'environnement de développement

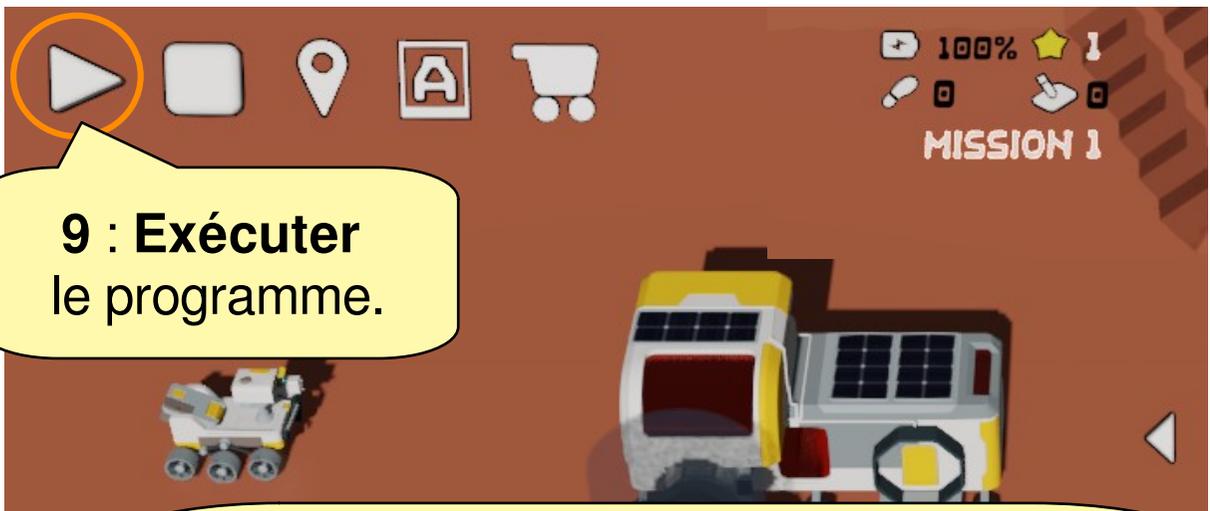


```
7 # @project: Ropy (Blender-EduTech)
8 #####
9
10 #####
11 # Initialisation du niveau :
12 # Niveau 1 : Les premiers pas de Ropy
13 # Niveau 2 : Ma première fonction
14 # Niveau 3 : Sécuriser Ropy
15 # Niveau 4 : Partir au bout du monde
16 # Niveau 5 : Faire face à l'inconnu
17 # Niveau 6 : Se
18 #####
19
20 #####
21 # Fonctions
22 #####
23
24 #####
25 # Commandes
26 #####
27
28 def commandes():
29
30     rp_gauche()
31     rp_avancerr()
32     rp_avancer()
33     rp_avancer()
34     rp_avancer()
35
36     rp_fin() # A garder
37
38 #####
```

7 : Écrire le code Python.

8 : Sauvegarder le fichier  
**Attention !**

Toujours sauvegarder le fichier avant son exécution.



9 : Exécuter le programme.

10 : Si votre code a une erreur, la console indique où elle se trouve. Ici la ligne 31 contient une commande indéfinie : '**rp\_avancerr**'.

```
Fichier Édition Affichage T
***** Module rp_cmd
rp_cmd.py:31: error (E0602, undefined-variable, commandes) Undefined variable 'rp_avancerr'
-----
Your code has been rated at 4.44/10 (previous run: 10.00/10, -5.56)
```

# Contenu du fichier rp\_cmd.py



Le fichier `rp_cmd.py` comporte 4 sections.

```
import time
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 : Ma première fonction
# Niveau 3 : Sécuriser Ropy
# Niveau 4 : Partir au bout du monde
# Niveau 5 : Faire face à l'inconnu
# Niveau 6 : Se rendre utile
#####

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

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

def commandes():
    → rp_gauche()
      rp_avancer()
      rp_avancer()
      rp_avancer()
      rp_avancer()

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

def cycle():
    commandes()
    rp_fin()

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

Le code doit être indenté  
(décalé sur la droite) avec  
la touche Tab

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

} **Fonctions** : section pour le codage de  
**vos fonctions**  
C'est votre outillage, à garder  
à travers les missions !

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

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



# Mission 2 - Ma première fonction

## Création d'une fonction



**Objectif 2** : Aller à la mission 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  
#####
```

# Mission 3 – Apprendre le danger

## Structure conditionnelle (si, alors, sinon)



**Objectif 3.1** : À la mission 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.





# Mission 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_nbpas (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.

angle est  
l'argument

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

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

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

# Mission 5 - Faire face à l'inconnu

## Structure itérative - boucle indéfinie (tant que)



**Objectif 5** : Aller à la mission 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  
#####
```





# Référence du langage de programmation de Ropy



## Instructions de base (rp\_\*):

- Avancer : `rp_avancer()`
- Reculer : `rp_reculer()`
- Tourner à gauche : `rp_gauche()`
- Tourner à droite : `rp_droite()`
- Marquer la case : `rp_marquer()`
- Détection d'un obstacle: `rp_detect()`
  - retourne **True** si il y a un obstacle
  - retourne **False** si il n'y a pas d'obstacle

## Instructions de base à créer (mrp\_\*):

- Avancer amélioré (marquage et sécurisation) : `mrp_avancer()`
- Avancer d'un nombre de pas : `mrp_avancer_nbpas(nb)`
- Avancer jusqu'à un obstacle : `mrp_avancer_mur()`

## Instructions avancées à créer (mrp\_\*):

- Aller à l'origine du balayage : `mrp_depart()`
- Faire un allée-retour : `mrp_aller_retour()`
- Faire un carré : `mrp_carre(nb_pas)`