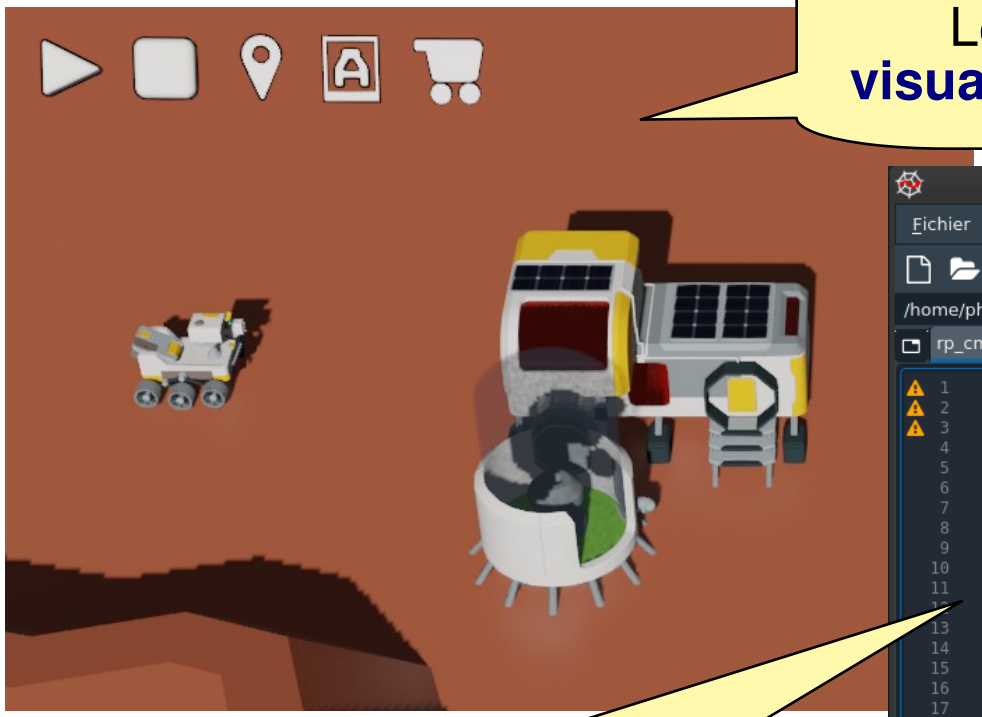
SNT

Introduction à la programmation avec Ropy

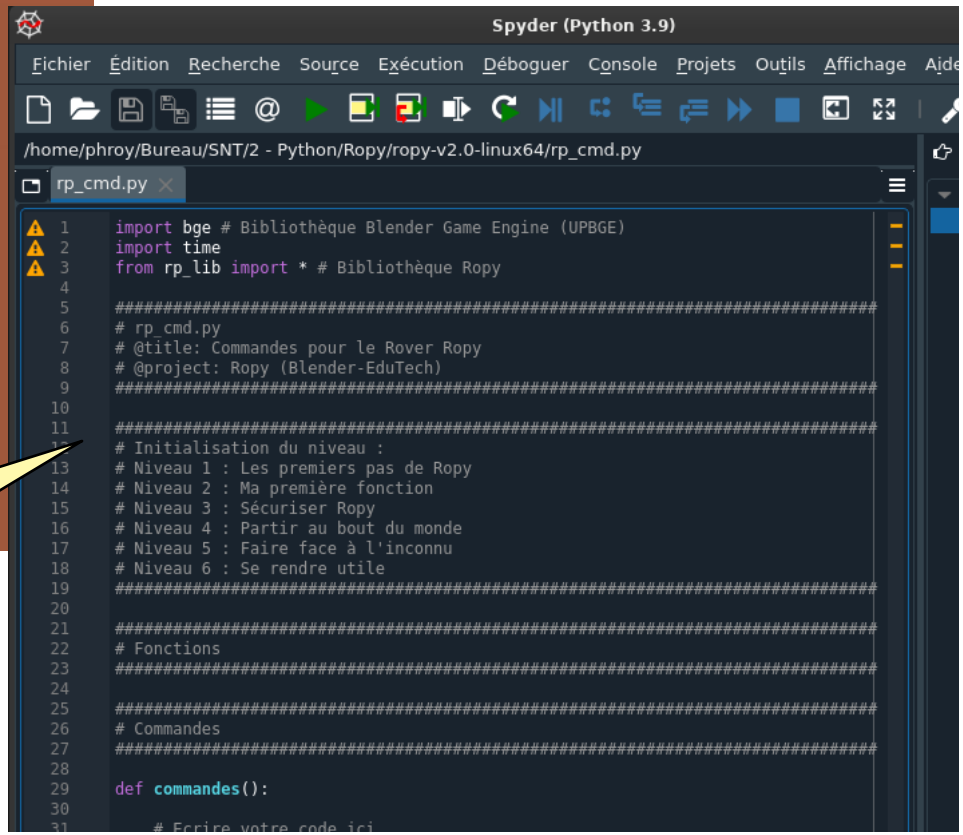
Présentation de Ropy et de son environnement de programmation

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.



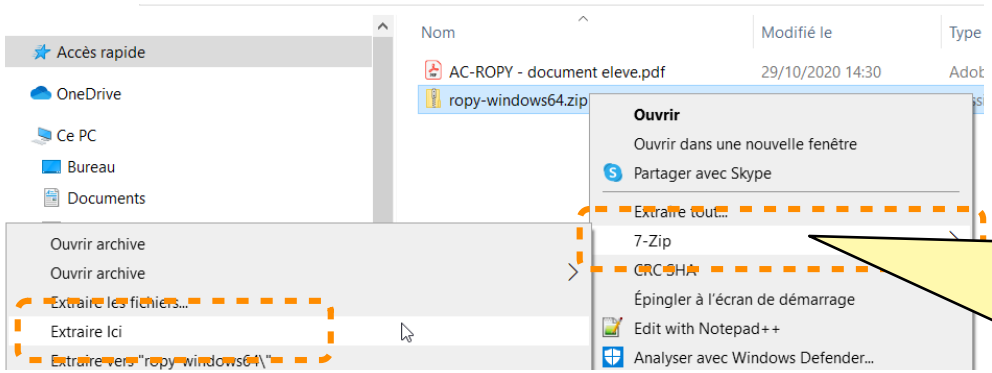
Le **simulateur** permet de **visualiser l'évolution du Rover**.

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



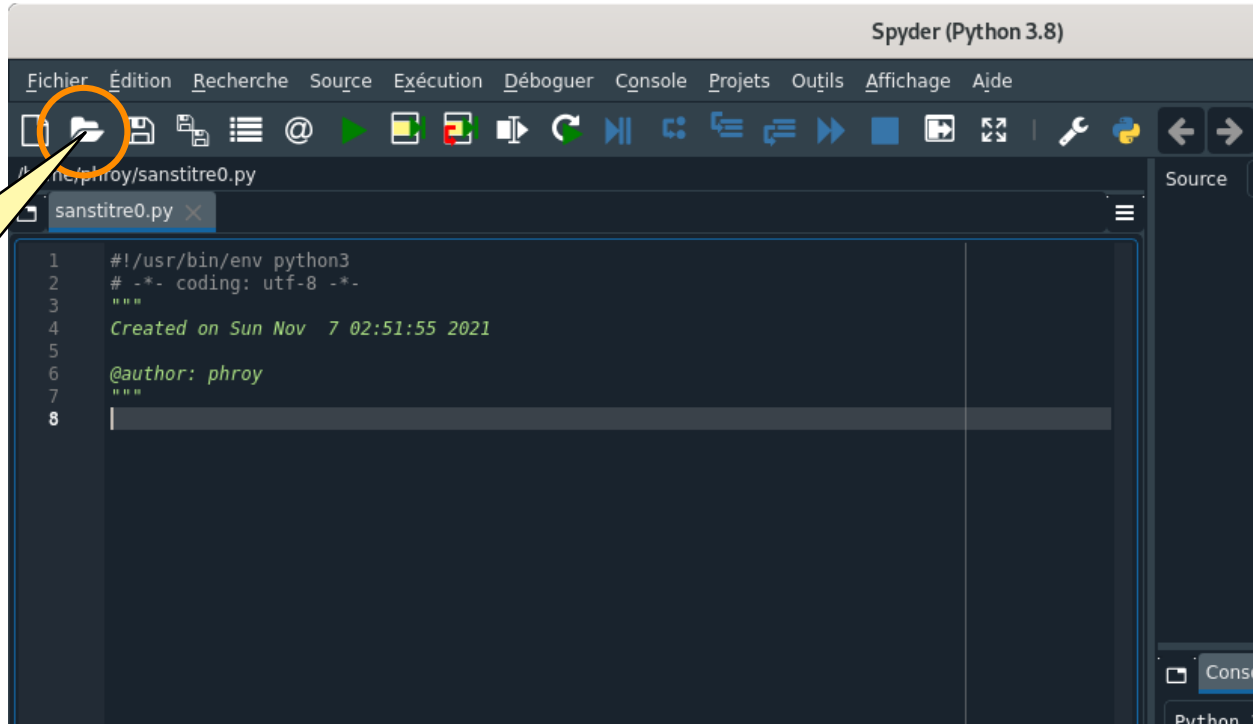
Éditer le programme avec Spyder

Ouvrir le fichier rp_cmd.py



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

2 : Lancer le Logiciel **Spyder**.



3: Ouvrir le fichier Python à éditer **rp_cmd.py** (Ropy commandes) présent dans le répertoire ropy.

Éditer le programme avec Spyder

Exécution du programme

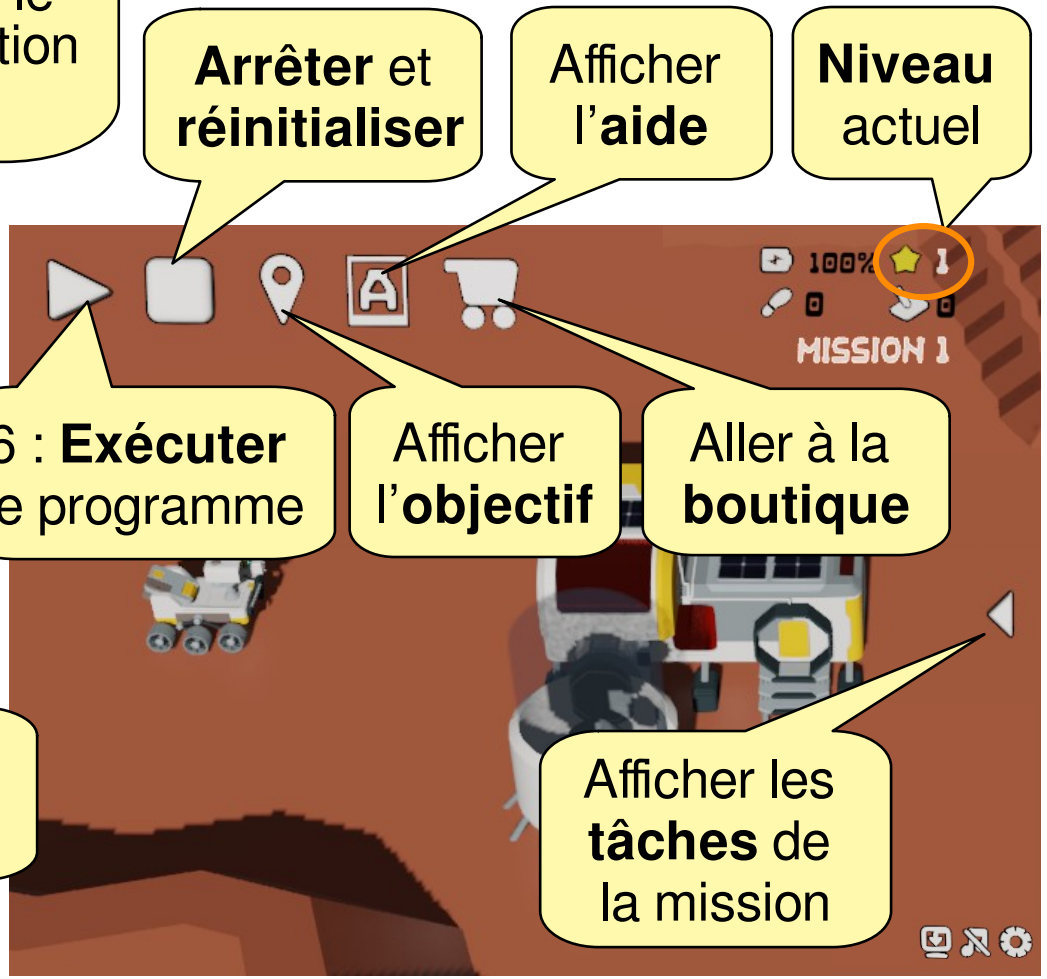
5 : **Sauvegarder** le fichier

Attention !

Toujours sauvegarder le fichier avant son exécution avec le simulateur.

4 : **Écrire** le code Python

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



Arrêter et **réinitialiser**

Afficher l'**aide**

Niveau actuel

6 : **Exécuter** le programme

Afficher l'**objectif**

Aller à la **boutique**

Afficher les **tâches** de la mission

Contenu du fichier rp_cmd.py

Le fichier **rp_cmd.py** comporte 4 sections.

```
import bge # Bibliothèque Blender Game Engine (UPBGE)
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()

    rp_fin() # A garder


##############################################################################
# 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()
```

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**

Commandes : section pour le codage des commandes du robot

La commande `rp_fin()` est à conserver.

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

Mission 1 – Les premiers pas de Ropy

Instruction et structure linéaire

Objectif 1 : Il faut aider **Ropy** à sortir du son emplacement et atteindre la case à l'est de la station. Afin de visualiser le trajet, il faudra marquer les cases.

Vous avez à disposition plusieurs **commandes élémentaires** pour diriger **Ropy** :

- Avancer : `rp_avancer()`
- Tourner à gauche : `rp_gauche()`
- Tourner à droite : `rp_droite()`
- Marquer la case : `rp_marquer()`

Le « `rp_` » dans le nom des fonctions permet d'identifier les fonctions de **Ropy**.

Le programme comporte des lignes avec le caractère **#**. Le texte qui suit ce caractère **#** est un **commentaire**, cela sert à documenter le code.

Un commentaire n'est pas exécuté.

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

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 définie **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.

Niveau 3 – Apprendre le danger

Structure conditionnelle (si, alors, sinon)

Objectif 3.2 : Intégrer le test de sécurisation dans votre fonction `mrp_avancer()`. Avancer uniquement en cas d'absence d'obstacle.

En lançant Ropy avec le fichier « ropy.bat » (à la place de ropy.exe) une **console** apparaît en plus du simulateur .

Nous pouvons alors générer des **message dans la console** afin de faciliter la vérification et la correction (débogage) du programme.

```
print("Texte à afficher")
```

Il est possible d'afficher la valeur d'une variable. Par exemple, je souhaite afficher la valeur de la variable `nb_pas` accompagnée d'un texte explicatif.

```
print("Nombre de pas:",nb_pas)
```

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

Mission 4 – Partir au bout du monde

Structure itérative - boucle définie

Objectif 4.1 : Aller à la mission 4, **Ropy** est maintenant prêt pour l'aventure et donc atteindre une case éloignée. Pour un tel voyage, l'utilisation d'une boucle s'impose.

Une **structure itérative** (boucle) permet de répéter plusieurs fois les même instructions.
Une **boucle définie** (nombre de répétitions connue à l'avance) est gérée par un **compteur**.

`variable` est le compteur qui sera incrémenté à chaque itération (de 0 à **`n`**-1).

```
for variable in range(n):
    bloc_instructions
```

Par exemple : répéter 3 fois l'avancement d'un pas du robot :

```
for i in range(3):
    rp_avancer()
```

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

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.

```
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
##########################################

_____

_____

_____

_____

_____
```

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 vrai**.

```
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
##########################################

_____________________________________

_____________________________________

_____________________________________

_____________________________________
```

Niveau 6 – Se rendre utile

Objectif 6.1 : Afin d'analyser la roche sur une zone, **Ropy** doit y prélever des carottes. Pour effectuer les forages, **Ropy** doit passer sur toutes les cases.

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

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

Niveau 6 – Se rendre utile … certes, mais avec classe !

Objectif 6.2 : **Ropy** est devenu esthète. C'est le même objectif, mais il faut parcourir le terrain en **colimaçon**.

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

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