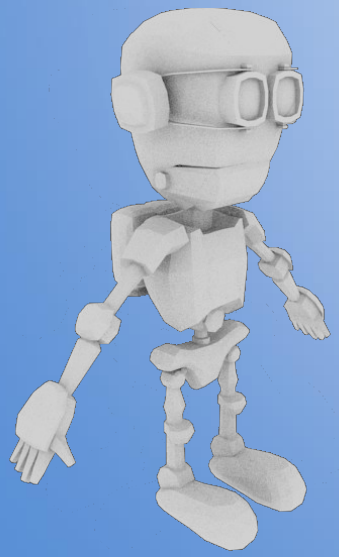
SNT

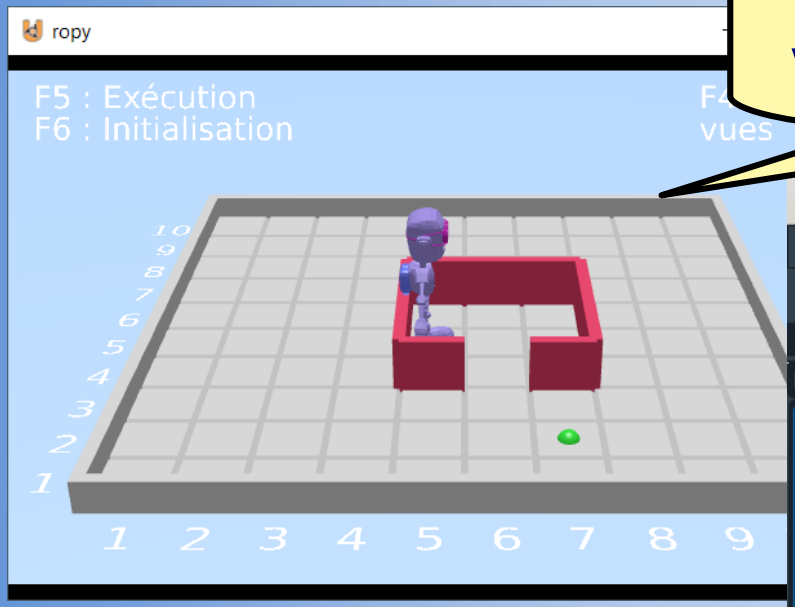
LYCÉE L'OISELET

Introduction à la programmation avec Ropy

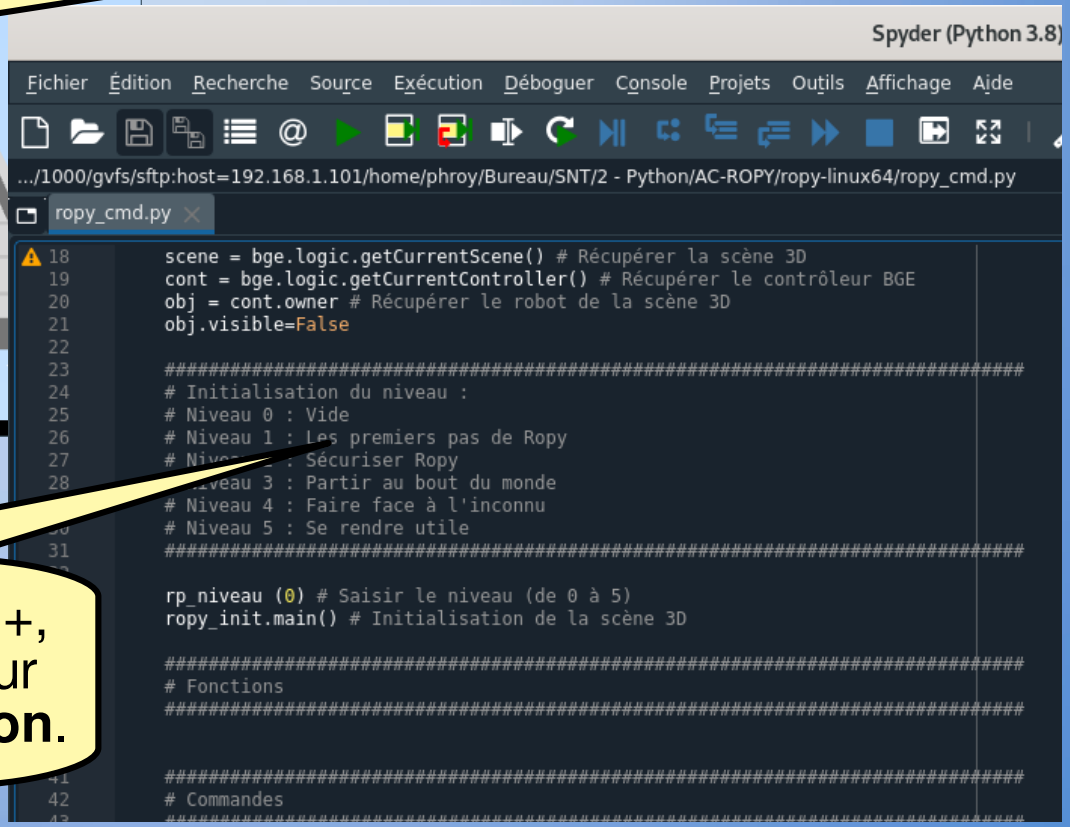


Présentation de Ropy et de son environnement de programmation

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



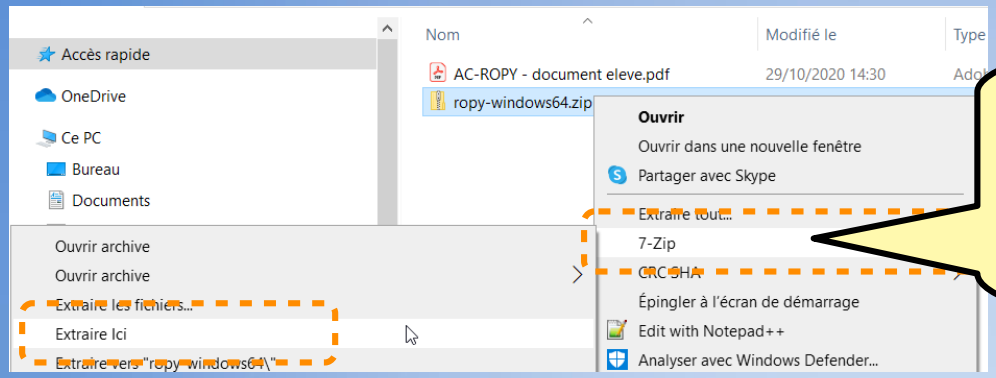
Le **simulateur** permet de **visualiser l'évolution du robot**.



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

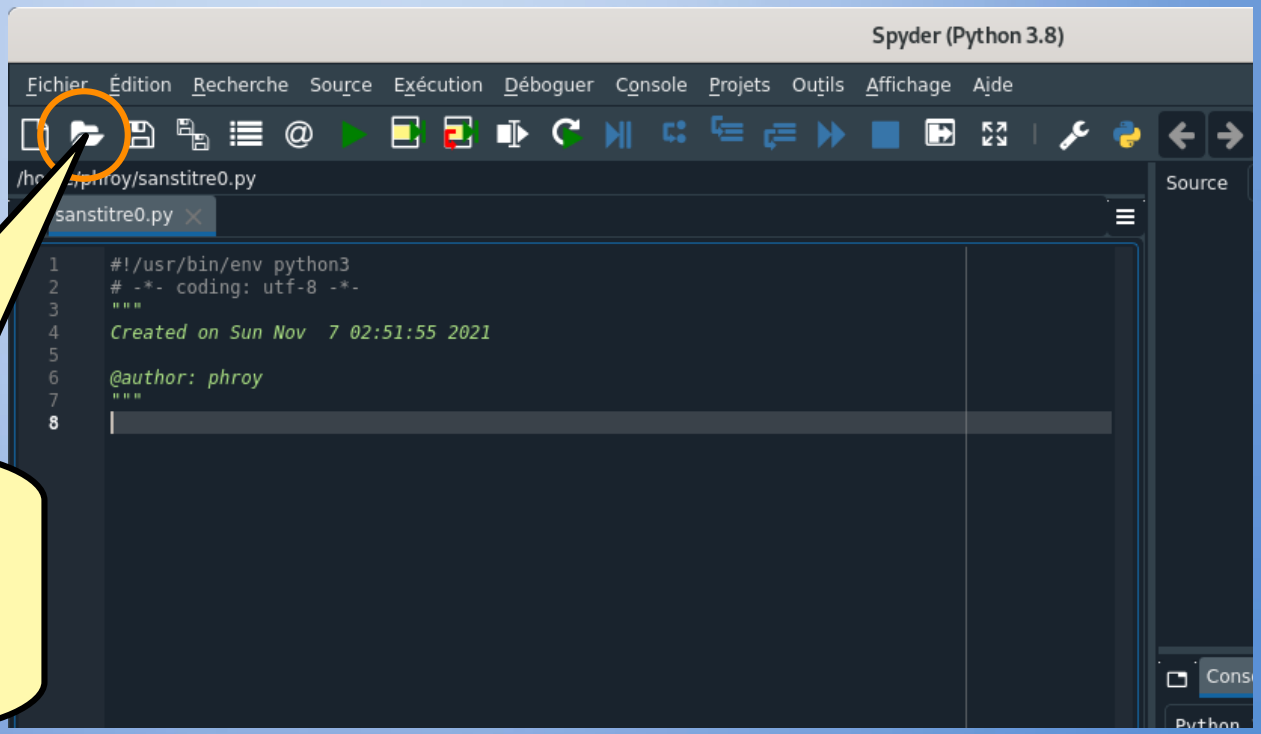
Éditer le programme avec Spyder

Ouvrir le fichier ropy_cmd.py



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

2 : Lancer le Logiciel **Spyder**.



3: Ouvrir le fichier Python à éditer **ropy_cmd.py** (Ropy commandes).

Éditer le programme avec Spyder

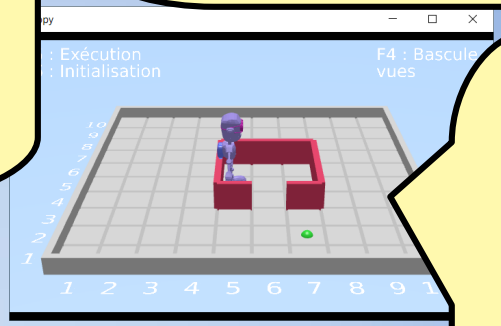
Exécution du programme

5 : **Sauvegarder** le fichier

Attention !

Toujours sauvegarder le fichier avant son exécution avec le simulateur.

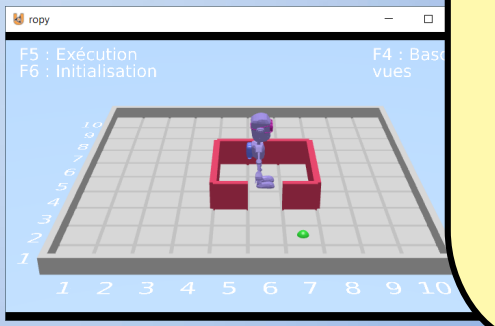
Le simulateur est le programme **ropy.exe**

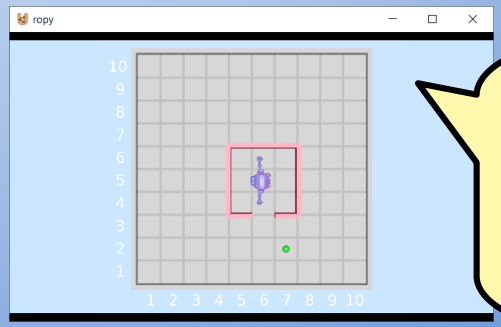


6 : **Exécuter** le programme

F6 : Revenir à l'état initial

F5 : Exécution de votre programme.





F4 (bascule) : Permet de changer de vue

4 : **Écrire** le code Python

```
19      cont = bge.
20      obj = cont.
21      obj.visible=Fa
22
23      ##########################################################################
24      # Initialisation du niveau :
25      # Niveau 0 : Vide
26      # Niveau 1 : Les premiers pas de Ropy
27      # Niveau 2 : Sécuriser Ropy
28      # Niveau 3 : Partir au bout du monde
29      # Niveau 4 : Faire face à l'inconnu
30      # Niveau 5 : Se rendre utile
31      ##########################################################################
32
33      rp_niveau (0) # Saisir le niveau (de 0 à 5)
34      ropy_init.main() # Initialisation de la scène 3D
35
36      ##########################################################################
37      # Fonctions
38      ##########################################################################
39
40
41      ##########################################################################
42      # Commandes
43      ##########################################################################
44
45      rp_avancer()
46      rp_avancer()
47      rp_avancer()
48      rp_avancer()
49
50
51
```

Contenu du fichier ropy_cmd.py

Le fichier **ropy_cmd.py** comporte 5 sections.

```
import bge # Bibliothèque Blender Game Engine (BGE)
import math # Bibliothèque Math
from ropy_lib import * # Bibliothèque Ropy
import ropy_init # Initialisation du robot Ropy

################################################################################
# ropy_cmd :
# @title: Commandes du Robot Ropy
# @project: Ropy (RobotProg pour Python)
################################################################################

def main():

    ################################################################################
    # Récupérer les objets du moteur 3D (BGE) << NE PAS MODIFIER CETTE SECTION >>
    ################################################################################

    scene = bge.logic.getCurrentScene() # Récupérer la scène 3D
    cont = bge.logic.getCurrentController() # Récupérer le contrôleur BGE
    obj = cont.owner # Récupérer le robot de la scène 3D
    obj.visible=False

    ################################################################################
    # Initialisation du niveau :
    # Niveau 0 : Vide
    # 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
    ################################################################################

    rp_niveau (0) # Saisir le niveau (de 0 à 5)
    ropy_init.main() # Initialisation de la scène 3D

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


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

    rp_avancer()
    rp_avancer()
    rp_avancer()
    rp_avancer()
```

Import des bibliothèques

Récupération des objets du moteur 3D BGE
(Blender Game Engine)
Ne pas modifier cette section

Initialisation du niveau

Vous devez entrer le niveau avec la commande
`rp_niveau(numero_niveau)`

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

Commandes : section pour le codage des commandes du robot

Niveau 1 – Les premiers pas de Ropy

Instruction et structure linéaire

Objectif 1.1 : Il faut aider **Ropy** à sortir du local et atteindre la case (7,2). 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**.

Afin de mieux comprendre le programme, nous pouvons le documenter. Il suffit d'ajouter le caractère **#** avant le commentaire.

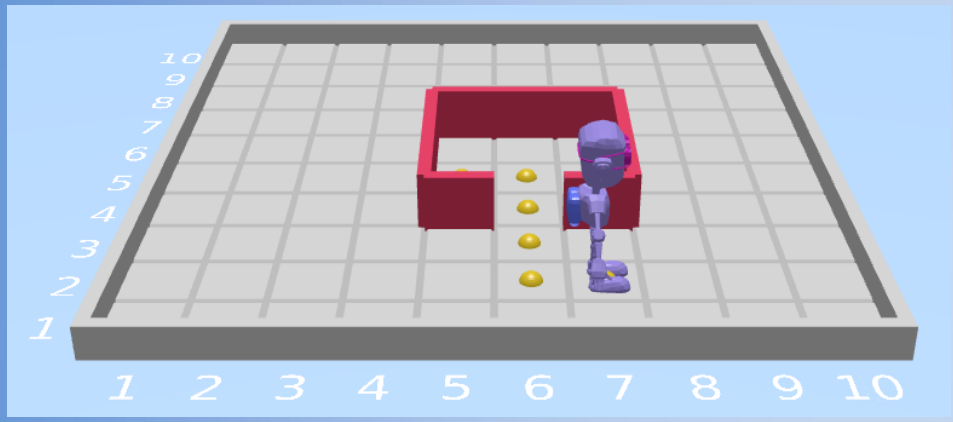
```
instruction_1 # Commentaire
```

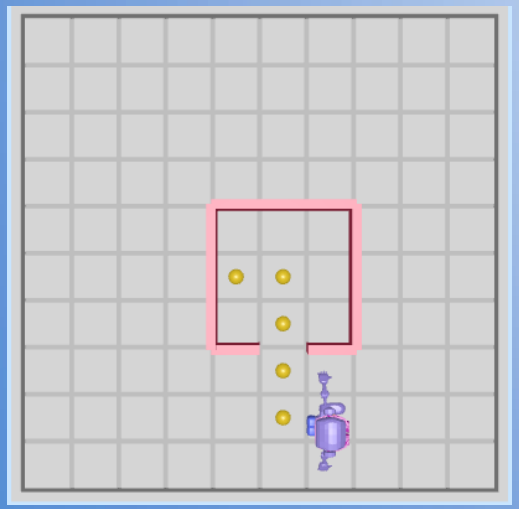
```
##########################################
# Commandes
##########################################
```

Niveau 1 – Les premiers pas de Ropy

Instruction et structure linéaire

Objectif 1.1 : Il faut aider **Ropy** à sortir du local et atteindre la case (7,2). Afin de visualiser le trajet, il faudra marquer les cases.





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

rp_marquer()
rp_avancer()
rp_marquer()
rp_droite()
rp_avancer()
rp_marquer()
rp_avancer()
rp_marquer()
rp_avancer()
rp_marquer()
rp_gauche()
rp_avancer()
rp_marquer()
```

Niveau 1 – Les premiers pas de Ropy

Création d'une fonction

Objectif 1.2 : 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 des plus simple :

```
fonction_1(arguments)
```

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

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

_____

_____

_____
```

Niveau 1 – Les premiers pas de Ropy

Création d'une fonction

Objectif 1.2 : 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 des plus simple :

```
fonction_1(arguments)
```

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

def mrp_avancer():
    rp_avancer()
    rp_marquer()



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

rp_marquer()
mrp_avancer()
rp_droite()
mrp_avancer()
mrp_avancer()
mrp_avancer()
rp_gauche()
mrp_avancer()
```

Niveau 1 – Les premiers pas de Ropy

Création d'une fonction

Objectif 1.3 : **Ropy** ne sait pas reculer, vous allez donc créer la fonction **`mrp_reculer()`**. Pour tester cette nouvelle fonction en toute sécurité (sans les murs), vous pouvez vous mettre sur **le niveau 0** avec la fonction **`rp_niveau(niveau)`**.

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

rp_niveau(0) # Saisir la mission (de 0 à 5)
ropy_ini  ain() # Initialisation de la scène 3D
```

Garder votre fonction **`mrp_avancer()`**. Ne marquer que la nouvelle fonction.

`rp_niveau(niveau)` se trouve avant la définition de vos fonctions.

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

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

Niveau 1 – Les premiers pas de Ropy

Création d'une fonction

Objectif 1.3 : **Ropy** ne sait pas reculer, vous allez donc créer la fonction **`mrp_reculer()`**. Pour tester cette nouvelle fonction en toute sécurité (sans les murs), vous pouvez vous mettre sur **le niveau 0** avec la fonction **`rp_niveau(niveau)`**.

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

rp_niveau(0) # Saisir la mission (de 0 à 5)
ropy_ini  ain() # Initialisation de la scène 3D
```

Garder votre fonction **`mrp_avancer()`**. Ne marquer que la nouvelle fonction.

`rp_niveau(niveau)` se trouve avant la définition de vos fonctions.

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

def mrp_reculer():
    rp_droite()
    rp_droite()
    mrp_avancer()
    rp_droite()
    rp_droite()
```

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

mrp_reculer()
```

Niveau 2 – Apprendre le danger

Structure conditionnelle (si, alors, sinon)

Objectif 2.1 : Aller au niveau 2, provoquer une collision avec un mur 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)

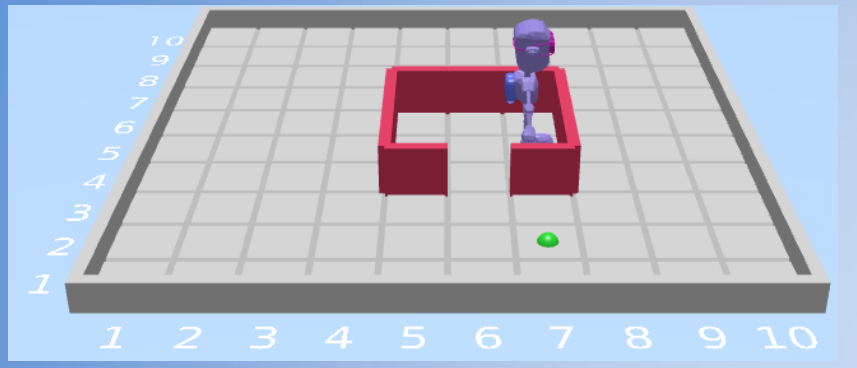
La fonction pour **détecter un mur** est : `rp_detect_mur()`. La fonction retourne `True` si il a un mur et `False` si il n'y a pas de mur.

Niveau 2 – Apprendre le danger

Structure conditionnelle (si, alors, sinon)

```
Avancer case (X,Y) -> case (X,Y) : 6 , 5 -> 7 , 5
Avancer case (X,Y) -> case (X,Y) : 7 , 5 -> 8 , 5
Bling, blang ... L'aventure de Ropy s'arrête donc ici ..
```

La console nous informe de la destruction du robot !




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

_____

_____

_____

_____

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

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

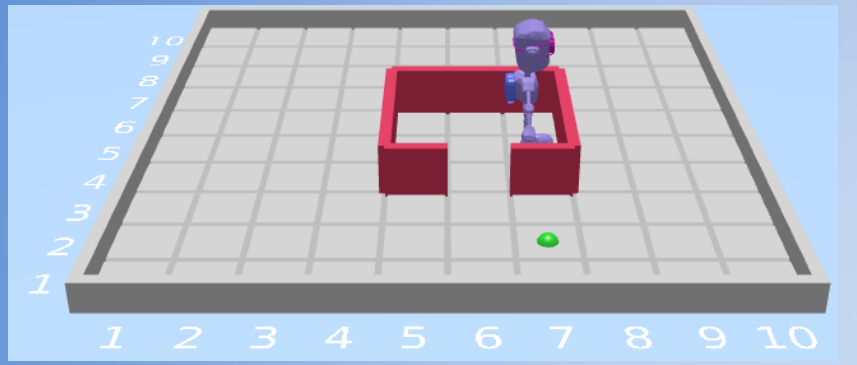
- a `==` b : a est égal à b
- a `!=` b : a est différent de b
- a `<` b : a est strictement inférieur à b
- a `<=` b : a est inférieur ou égal à b
- a ==b `and` c==d : les deux conditions doivent être vrai (fonction ET)
- a ==b `or` c==d : une des deux conditions doit être vrai (fonction OU)

Niveau 2 – Apprendre le danger

Structure conditionnelle (si, alors, sinon)

```
Avancer case (X,Y) -> case (X,Y) : 6 , 5 -> 7 , 5
Avancer case (X,Y) -> case (X,Y) : 7 , 5 -> 8 , 5
Bling, blang ... L'aventure de Ropy s'arrête donc ici ..
```

La console nous informe de la destruction du robot !



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

mrp_avancer()
mrp_avancer()
if rp_detect_mur()== False:
   mrp_avancer()
```

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

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)

Niveau 2 – Apprendre le danger

Affichage d'un message vers la console

Objectif 2.2 : Intégrer le test de sécurisation dans votre fonction **`mrp_avancer()`**. Avancer uniquement en cas d'absence de mur.

Nous pouvons 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
##########################################
```

Niveau 2 – Apprendre le danger

Affichage d'un message vers la console

Objectif 2.2 : Intégrer le test de sécurisation dans votre fonction **`mrp_avancer()`**. Avancer uniquement en cas d'absence de mur.

Nous pouvons 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
##########################################

def mrp_avancer():
  if rp_detect_mur()==False:
    rp_avancer()
    rp_marquer()
  else:
    print("Il y a un mur !")
    print("Avance annulée.")
```

```
Détection de mur : voie libre : Position case (X,Y): 5 5
Avancer case (X,Y) -> case (X,Y) : 5 , 5 -> 6 , 5
Détection de mur : voie libre : Position case (X,Y): 5 6
Avancer case (X,Y) -> case (X,Y) : 6 , 5 -> 7 , 5
Détection de mur : mur devant ! : Position case (X,Y):  5 7
Il y a un mur ! Avance annulée.
```

La console nous informe de l'annulation du mouvement.

Niveau 3 – Partir au bout du monde

Structure itérative - boucle définie

Objectif 3.1 : Aller au niveau 3, **Ropy** est maintenant prêt pour l'aventure soit atteindre la case (10,10). Pour un tel voyage, l'utilisation de 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, partant de 0 pour atteindre **`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
##########################################
```

_____

_____

_____

_____

_____

_____

_____

_____

_____

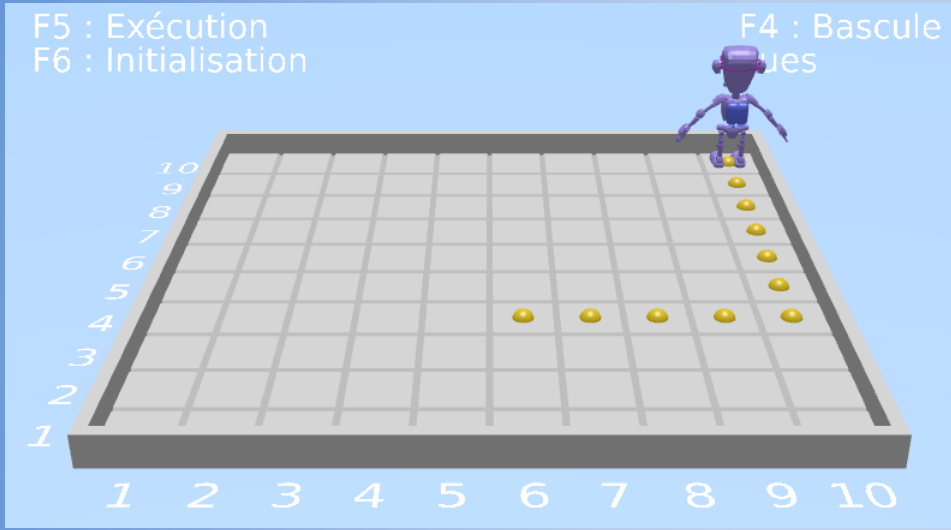
_____

_____

Niveau 3 – Partir au bout du monde

Structure itérative - boucle définie

Objectif 3.1 : Aller au niveau 3, **Ropy** est maintenant prêt pour l'aventure soit atteindre la case (10,10). Pour un tel voyage, l'utilisation de boucle s'impose.




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

for i in range(5):
   mrp_avancer()
rp_gauche()
for i in range(6):
   mrp_avancer()
```

Niveau 3 – Partir au bout du monde

Passage d'argument (dans une fonction)

Objectif 3.2 : 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 3 – Partir au bout du monde

Passage d'argument (dans une fonction)

Objectif 3.2 : 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
##########################################

def mrp_avancer_pas(nb_pas):
    for i in range(nb_pas):
        mrp_avancer()

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

mrp_avancer_pas(5)
rp_gauche()
mrp_avancer_pas(6)
```

Attention ! double indentation

Niveau 4 – Faire face à l'inconnu

Structure itérative - boucle indéfinie (tant que)

Objectif 4 : Aller au niveau 4, **Ropy** doit toujours atteindre la case (10,10), mais son lieu de départ change à chaque fois. Pour pallier à l'aléatoire, il faut créer une fonction qui permet d'atteindre un mur : **`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 4 – Faire face à l'inconnu

Structure itérative - boucle indéfinie (tant que)

Objectif 4 : Aller au niveau 4, **Ropy** doit toujours atteindre la case (10,10), mais son lieu de départ change à chaque fois. Pour pallier à l'aléatoire, il faut créer une fonction qui permet d'atteindre un mur : **`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
##########################################

def mrp_avancer_mur():
  while rp_detect_mur()==False:
    mrp_avancer()


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

mrp_avancer_mur()
rp_gauche()
mrp_avancer_mur()
```

Niveau 5 – Se rendre utile

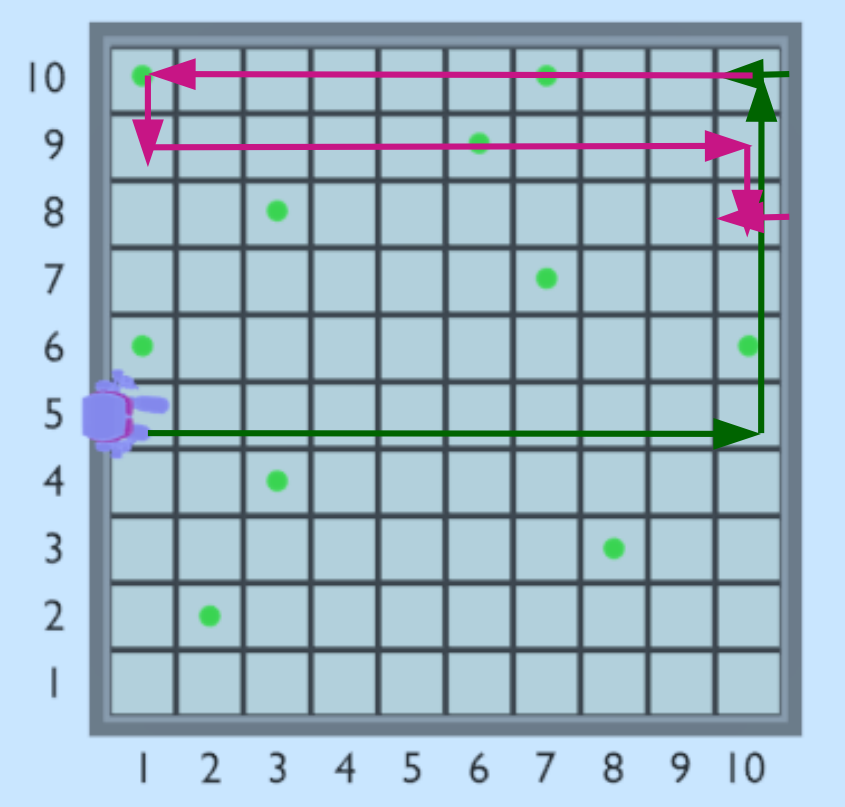
Objectif 5.1 : Le terrain est maintenant couvert de déchets. Pour nettoyer le terrain, **Ropy** doit pouvoir passer sur toutes les cases.

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

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

Niveau 5 – Se rendre utile

Objectif 5.1 : Le terrain est maintenant couvert de déchets. Pour nettoyer le terrain, **Ropy** doit pouvoir passer sur toutes les cases.




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

# Aller à la case (10,10)
def mrp_depart():
  mrp_avancer_mur()
  rp_gauche()
  mrp_avancer_mur()
  rp_gauche()

# Faire un aller-retour
def mrp_aller_retour():
  mrp_avancer_mur()
  rp_gauche()
  mrp_avancer()
  rp_gauche()
  mrp_avancer_mur()
  rp_droite()
  mrp_avancer()
  rp_droite()
```

Niveau 5 – Se rendre utile

Objectif 5.1 : Le terrain est maintenant couvert de déchets. Pour nettoyer le terrain, **Ropy** doit pouvoir passer sur toutes les cases.

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

mrp_depart()

for i in range (5):
  mrp_aller_retour()
```

Le « avancer » n'est pas exécuté grâce à l'anti-collision de **`mrp_avancer()`**

Niveau 5 – Se rendre utile … certes, mais avec classe !

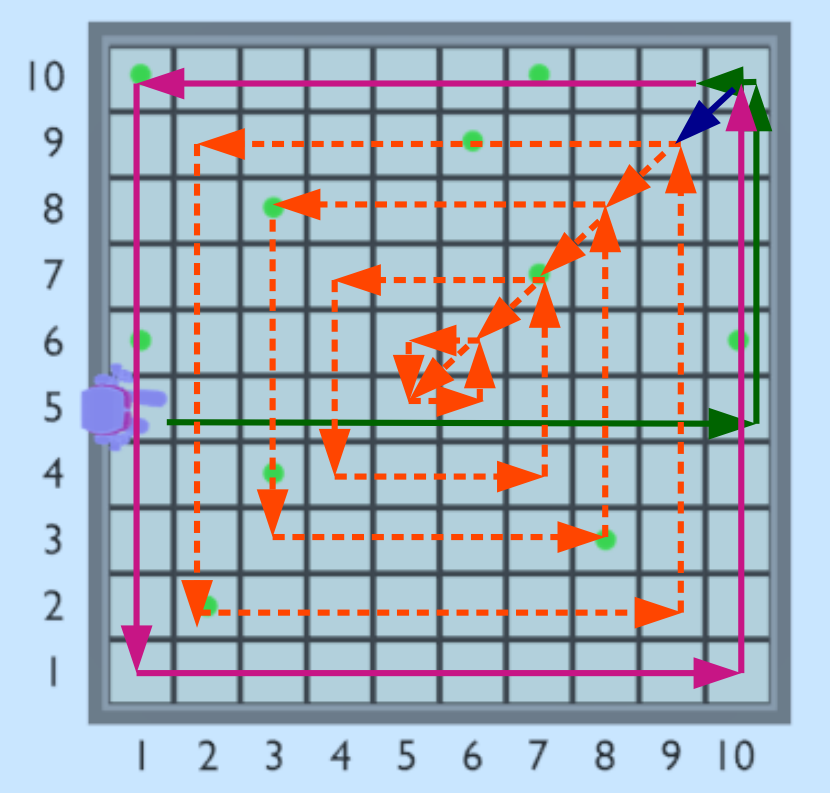
Objectif 5.2 : **Ropy** est devenu esthète. C'est le même objectif, mais il faut parcourir le terrain en **colimaçon**.

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

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

Niveau 5 – Se rendre utile … certes, mais avec classe !

Objectif 5.2 : **Ropy** est devenu esthète. C'est le même objectif, mais il faut parcourir le terrain en **colimaçon**.




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

# Faire un carre
def mrp_carre(nb_pas):
   for i in range (4):
      mrp_avancer_pas(nb_pas)
      rp_gauche()

# Avance d'une case
# en diagonale sud-ouest
def mrp_avancer_diag_so():
   rp_gauche()
   mrp_avancer()
   rp_droite()
   mrp_avancer()
```

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

mrp_depart()

nb_pas = 9
for i in range (5):
   mrp_carre(nb_pas)
   mrp_avancer_diag_so()
   nb_pas=nb_pas-2
```