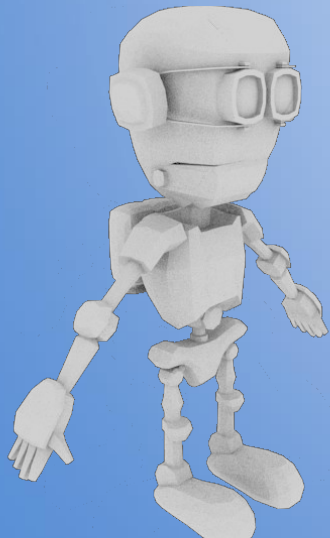
Séquence 5a

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

IT+I2D

LYCÉE L'OISELET

Introduction à la programmation Python avec Ropy



Niveau 1 – Les premiers pas de Ropy
Instruction et structure linéaire

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

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**.

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

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 : Créer la fonction **`mrp_avancer()`** regroupant avancer et marquer.

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

def mrp_avancer():
   rp_avancer()
   rp_marquer()



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

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

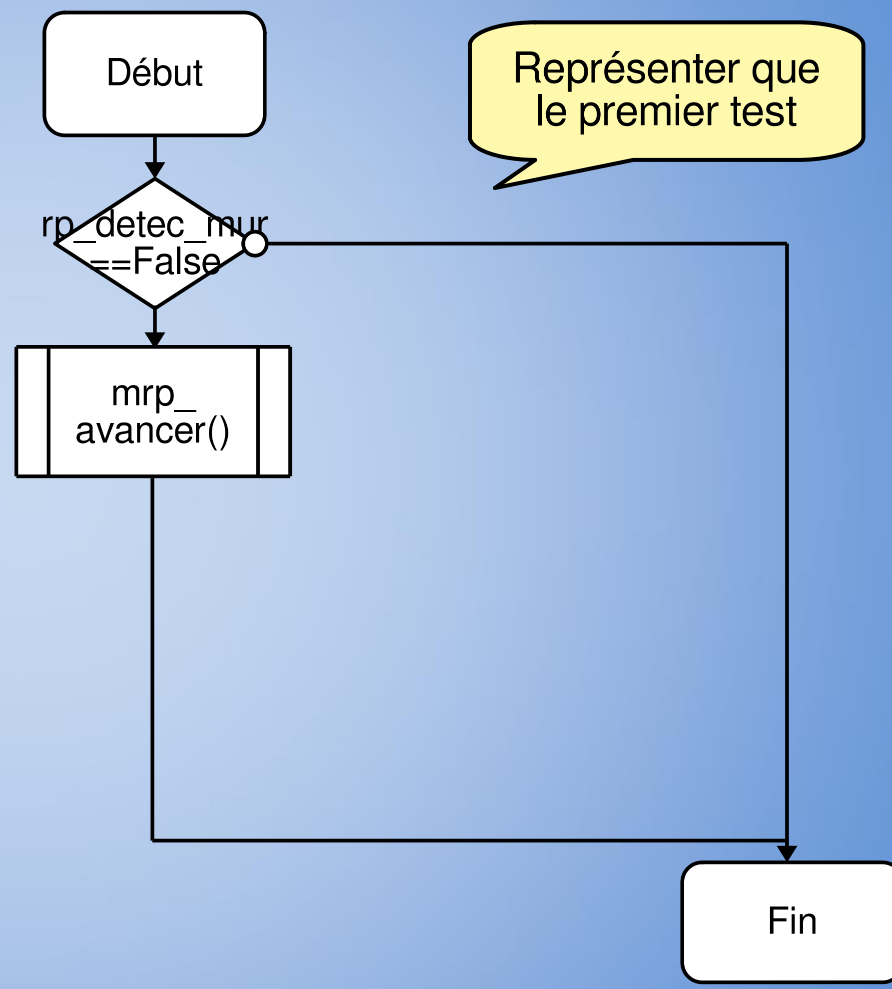
Objectif 1.3 : **Ropy** ne sait pas reculer, vous allez donc créer la fonction **`mrp_reculer()`**.

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

def mrp_reculer():
   rp_droite()
   rp_droite()
   rp_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 qui se passe. Il vous faut donc sécuriser l'avance du robot.

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

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

Niveau 2 – Apprendre le danger
Structure conditionnelle (si, alors, sinon)

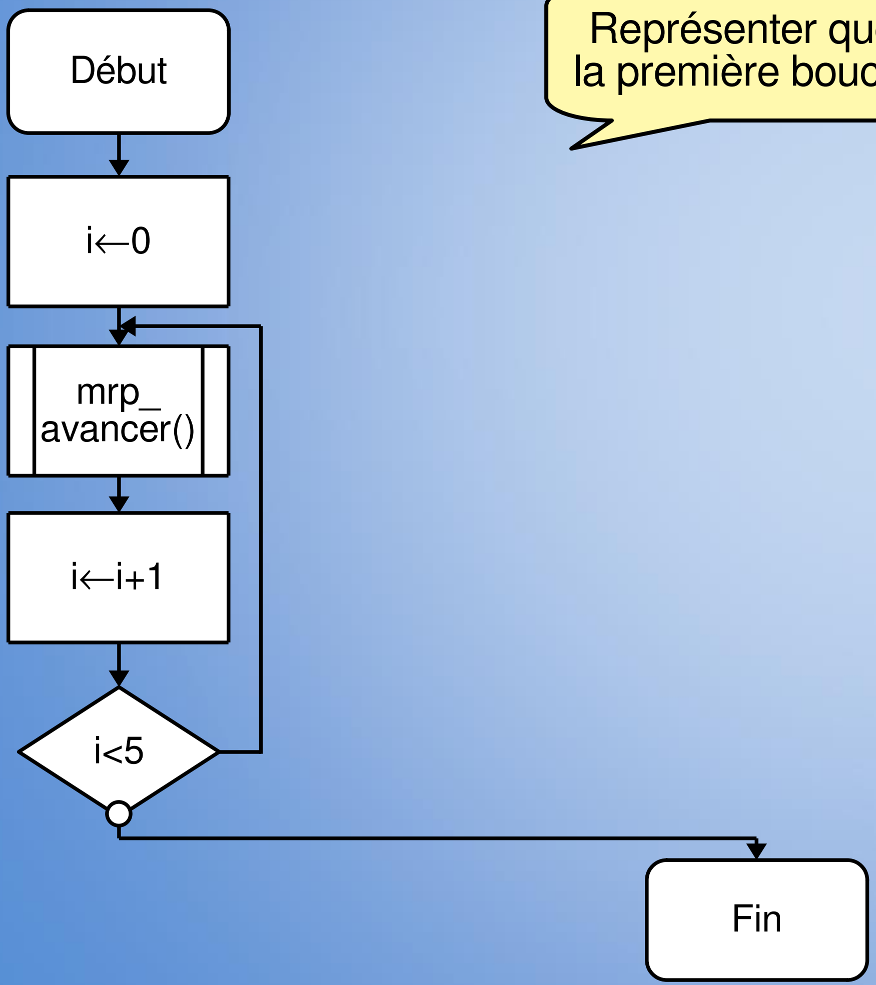
Objectif 2.2 : Intégrer le test de sécurisation dans votre fonction `mrp_avancer()`.

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

def mrp_avancer():
   if rp_detect_mur()==False:
      rp_avancer()
      rp_marquer()
```

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

mrp_avancer()
mrp_avancer()
mrp_avancer()
```

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.



Représenter que la première boucle

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

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

Niveau 3 – Partir au bout du monde
Structure itérative - boucle définie

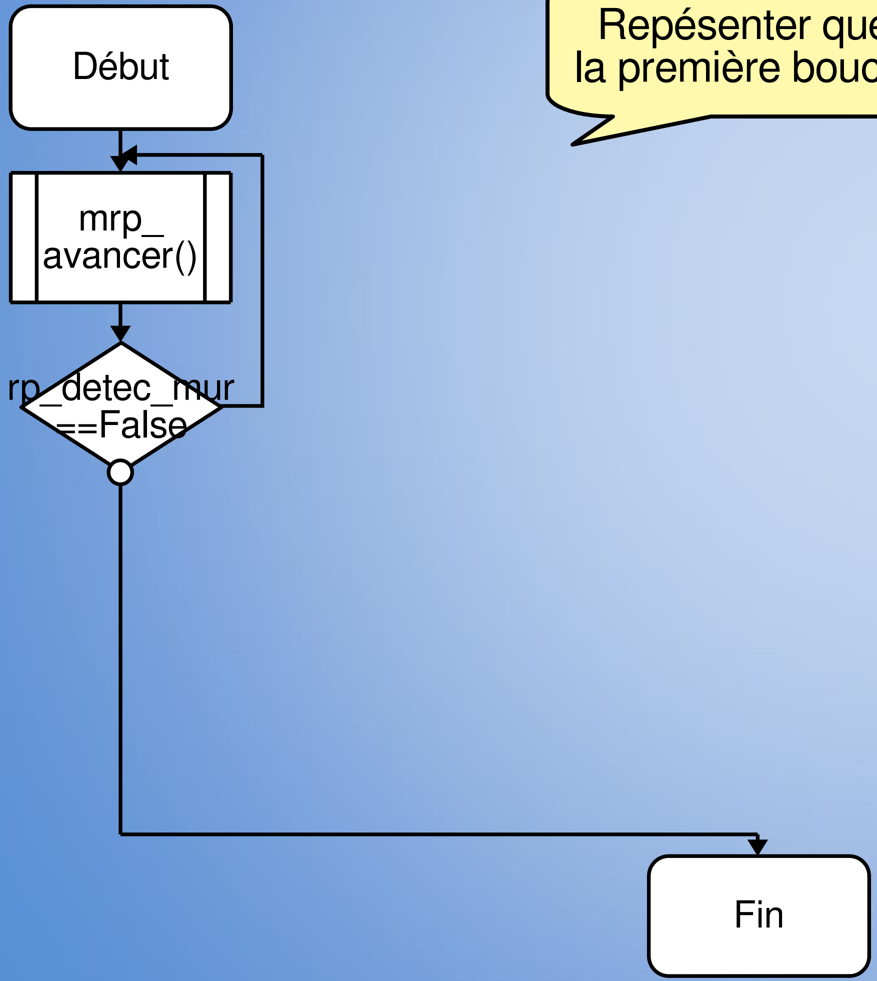
Objectif 3.2 : Afin d'enrichir notre bibliothèque, nous allons créer une fonction pour avancer d'un nombre de pas : **`mrp_avancer_pas(nb_pas)`**.

```python
#################################################
# Fonctions
#################################################

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

```python
#################################################
# Commandes
#################################################

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

Niveau 4 – Faire face à l’inconnu
Structure itérative - boucle indéfinie (tant que)

Objectif 4.1 : Aller au niveau 4, **Ropy** doit toujours atteindre la case (10,10), mais son lieu de départ change à chaque fois.



Repésenter que la première boucle

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

while rp_detect_mur()==False:
  mrp_avancer()
rp_gauche()
while rp_detect_mur()==False:
  mrp_avancer()
```

Niveau 4 – Faire face à l'inconnu
Structure itérative - boucle indéfinie (tant que)

Objectif 4.2 : Afin d'enrichir notre bibliothèque, nous allons créer une fonction qui permet d'atteindre un mur : **`mrp_avancer_mur()`**.

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

# 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()
```

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

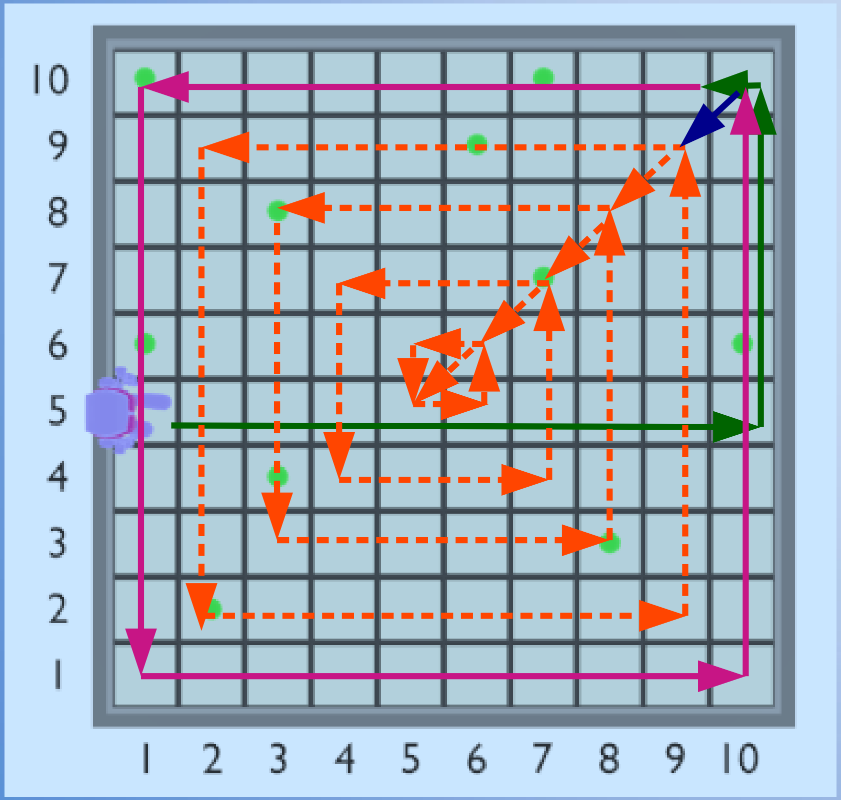
Niveau 5 – Se rendre utile

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

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

mrp_depart()

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

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()
```

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

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

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

mrp_depart()

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

Référence du langage de programmation de Ropy

Instructions de base (rp_*) :

- Avancer : `rp_avancer()`
- Tourner à gauche : `rp_gauche()`
- Tourner à droite : `rp_droite()`
- Marquer la case : `rp_marquer()`
- Détection d'un mur : `rp_detect_mur()`
  - retourne `True` si il y a un mur
  - retourne `False` si il n'y a pas de mur

Instructions de base à créer (mrp_*) :

- Avancer amélioré (marquage et sécurisation) : `mrp_avancer()`
- Avancer d'un nombre de pas : `mrp_avancer_pas(nb_pas)`
- Avancer jusqu'à un mur : `mrp_avancer_mur()`
- Reculer : `mrp_reculer()`

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)`