

➤ **Description de la situation :**

Il est possible de programmer un système dans un autre langage que le langage par blocs. C'est la solution privilégiée dans le milieu de l'industrie, où le langage Micro-python est couramment utilisé.

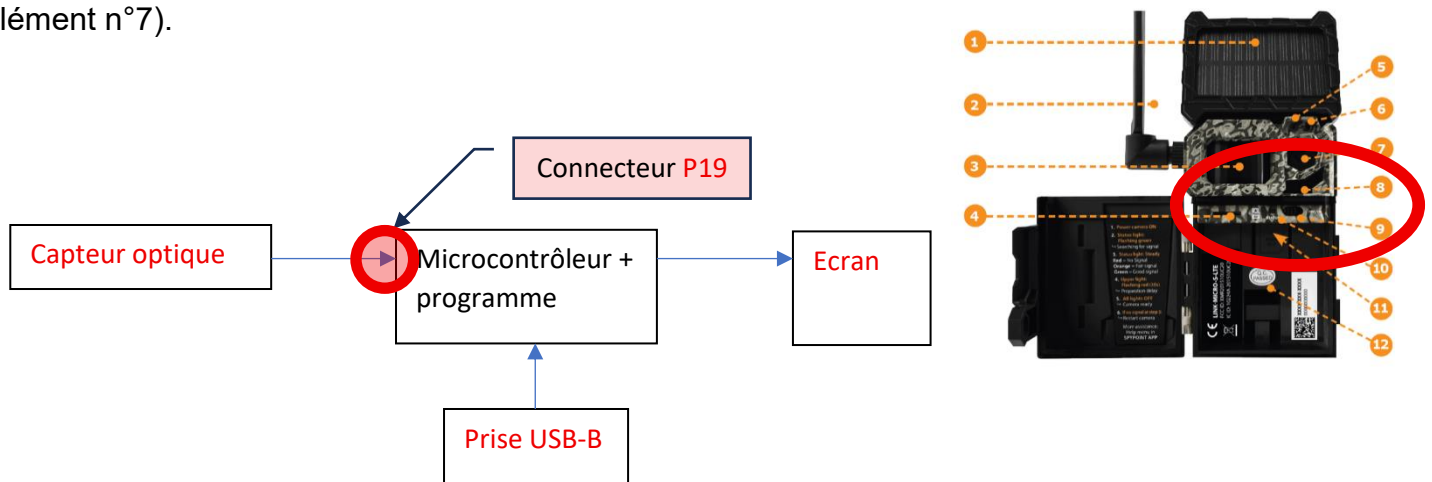
Un élève a commencé à programmer le système, mais cela ne fonctionne pas : certaines commandes sont manquantes dans le programme proposé.

➤ **Problématique :**

Comment programmer la carte microcontrôleur en mode texte (Micro Python) pour faire la reconnaissance d'un animal ?

➤ **Investigations :**

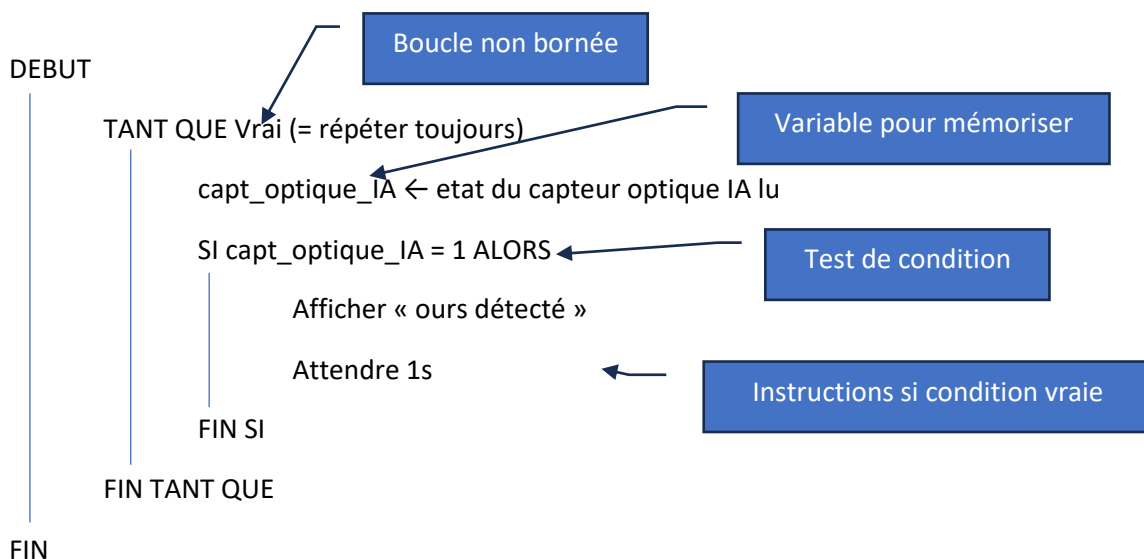
Lors de la séance 1, nous avons vu qu'un piège photographique utilisait une caméra (élément n°7).



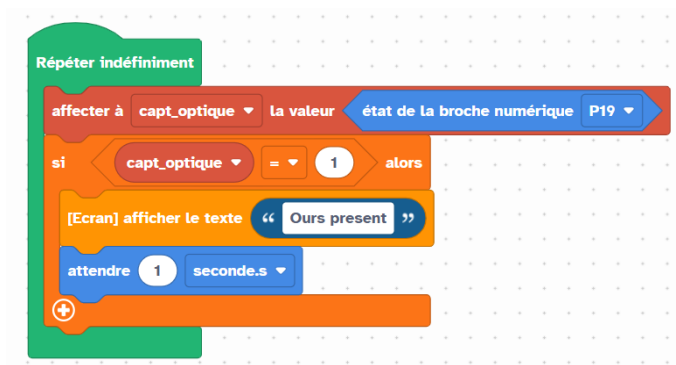
On a étudié une version simplifiée du système utilisant un capteur optique branché en P19.

On a étudié comment programmer le piège pour qu'il affiche « ours détecté » sur l'écran de la carte à chaque détection d'un ours par le système.

Le programme proposé est basé sur l'algorithme suivant :



La structure de programme suivante, rédigée en MicroPython, est associée au même algorithme que le programme en mode blocs.



```
1 from machine import *
2 from thingz import *
3 import utime
4
5 while True:
6     ?
7     if ? :
8         ?
9         ?
```

Pour la suite du travail, nous n'utiliserons que l'interface Python.

Lancer l'application Vittascience et démarrer un nouveau programme :

<https://fr.vittascience.com/galaxia/?mode=code&console=bottom&toolbox=vittascience>

En s'aidant de la structure ci-dessous, compléter le programme MicroPython :

```
1 from machine import *
2 from thingz import *
3 import utime
4
5 while True:
6     ?
7     if ? :
8         ?
9         ?
```

Tabulation à
conserver !

```
from machine import *
from thingz import *
import utime
while True:
```

```
.....
if .....:
.....
.....
```

Ajouter la bibliothèque « utime » à la ligne 3.

Compléter le programme principal avec les commandes textuelles manquantes.

Remplacer les « ? » par les commandes suivantes :

La commande **capt_optique = p19.value()** permet de mémoriser la valeur de l'entrée logique sur le connecteur P19 dans une variable nommée « *capt_optique* » par l'utilisateur.

Le test de condition **capt_optique == 1** permet de tester si le contenu de la variable *capt_optique* est égal à 1. Cette condition peut être vraie ou fausse.

La commande **print(texte)** permet d'afficher un texte ou des nombres sur l'écran de la carte Galaxia. Ici on remplacera texte par **'Ours present'**.

La commande **utime.sleep(1)** permet d'attendre 1s.

Utiliser le simulateur dans un premier temps pour valider le programme. Puis vérifier le fonctionnement du piège

➤ **Bilan de mes recherches :**

➤ **Pour aller plus loin**

Ajouter les commandes suivantes au programme précédent pour afficher le nombre de détections. Attention, chaque commande possède un emplacement bien précis.

nb_ours = 0 (# déclaration et initialisation d'une variable qui comptabilise le nombre d'ours présents)

nb_ours = nb_ours + 1 (# permet d'ajouter 1 à la variable nb_ours : on compte les ours)

print('Ours detecte(s)='+str(nb_ours)) (# permet d'afficher le nombre d'ours détectés)

Il est possible d'ajouter aussi la durée en secondes depuis le démarrage de la caméra afin de « dater » les détections.

Les commandes suivantes sont alors utilisées :

t0 = utime.ticks_ms() (# permet de mémoriser le nombre de millisecondes depuis le démarrage de la carte.)

print(str(round(utime.ticks_diff(utime.ticks_ms(), t0)/1e3))) # permet d'afficher la durée en seconde depuis le démarrage de la carte. On divise par 1000 = 10^3 pour passer des millisecondes aux secondes.

➤ **Bilan commun :**

Un test de condition s'écrit :

```
if capt_optique == 1:
    nb_ours = nb_ours + 1
    print('Ours detecte(s)='+str(nb_ours))
    utime.sleep(1)
```