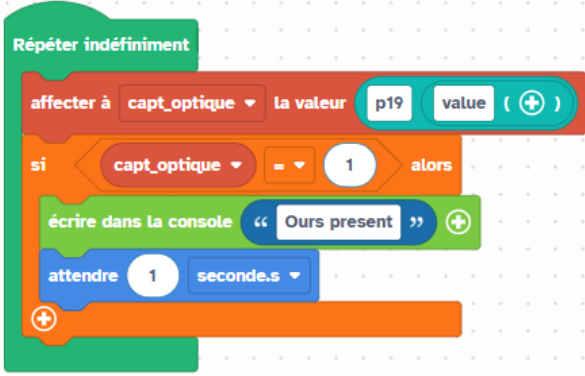


  MINISTÈRE DE L'ÉDUCATION NATIONALE ET DE LA JEUNESSE	Séance S3 DOCUMENT PROFESSEUR CORRECTION	 ACADÉMIE DE LYON <i>Liberté Égalité Fraternité</i>
CYCLE 4 Séance S3	Comment suivre un animal à distance ?	NIVEAU TROISIEME

➤ **Mise en situation :**



```

1 from machine import *
2 from thingz import *
3 import utime
4
5 while True:
6     capt_optique = p19.value()
7     if capt_optique == 1:
8         print('Ours present')
9         utime.sleep(1)
10
        
```

- Est-il possible de programmer un même algorithme dans différents langages ?

Oui, il y a ici 2 IDEs en mode blocs ou textes.

- Quels sont les langages proposés ?

On a le langage graphique par blocs et l'autre sous forme de texte (langage Python).

Pour comprendre le lien entre les langages, visionner la vidéo :

<https://www.youtube.com/watch?v=yP5KkHkv18A>

Executer
Ouvrir
Sauver
Insérer blocs
Traduire blocs

```

1 reponse = float(input('Combien font 7 x 8 ?'))
2 if (reponse == 56) :
3     print('Bravo !')
4 else :
5     print('Perdu !')
        
```

Executer
Ouvrir
Sauver
Effacer

Entrées-sorties

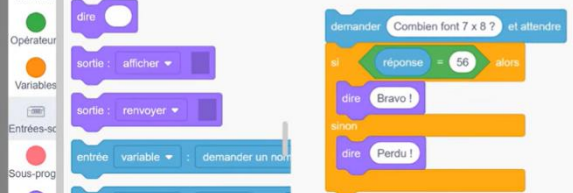
Contrôle

Opérateur

Variables

Entrées-sc

Sous-prog



➤ **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.

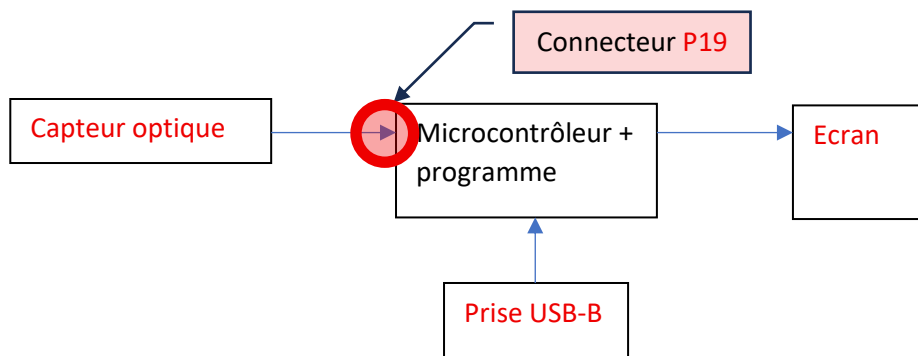
Un élève a commencé à programmer le système, mais cela ne fonctionne pas : certaines commandes sont manquantes dans le programme proposé.

➤ **Problématique :**

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

➤ **Investigations :**

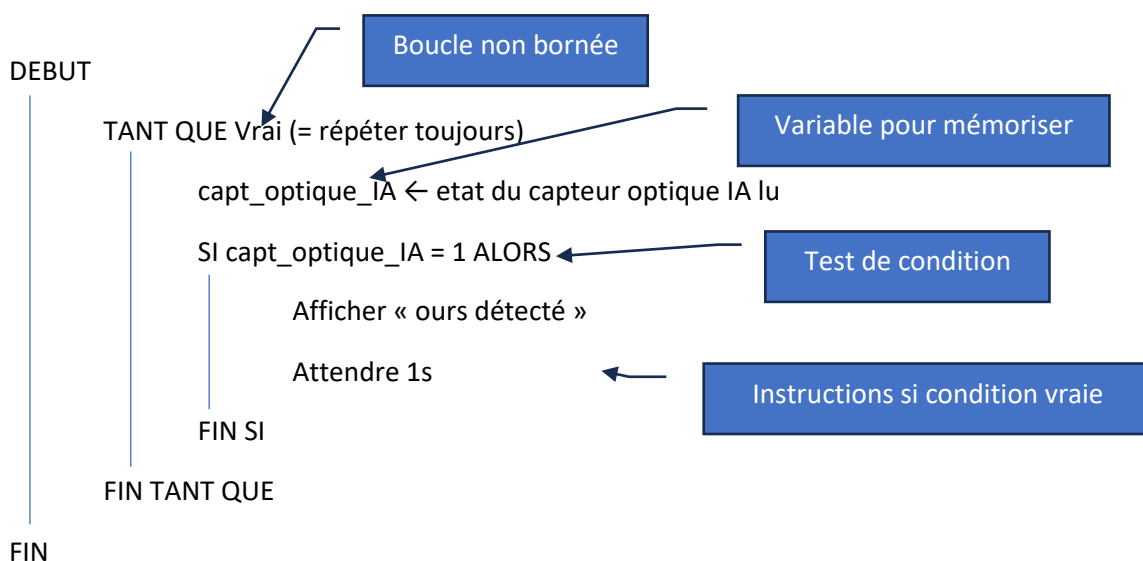
Lors de la séance 1, nous avons vu qu'un piège photographique utilisait une caméra (élément n°7).



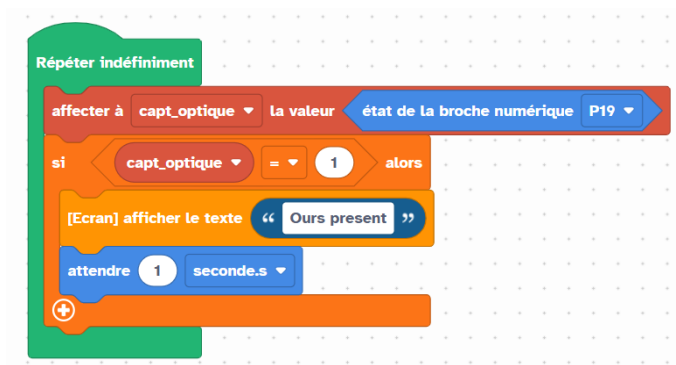
On a étudié une version simplifiée du système utilisant un capteur optique branché en P19.

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.

Bien remplacer les « ? » par ces commandes ! Ne pas supprimer les indentations (tabulations) !

Correction du programme :

Tabulation à
conserver !

```
1 from machine import *
2 from thingz import *
3 import utime
4
5 while True:
6     capt_optique = p19.value()
7     if capt_optique == 1:
8         print('Ours present')
9         utime.sleep(1)
```

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

Correction :

```
1 from machine import *
2 from thingz import *
3 import utime
4
5 p19 = Pin(13, Pin.IN)
6
7 while True:
8     capt_optique = p19.value()
9     if capt_optique == 1:
10         print('Ours present')
11         utime.sleep(1)
12
```

➤ Bilan de mes recherches :

Il est possible de concevoir un programme avec un IDE dans un autre langage, tel que le Micro Python. Son écriture impose de connaître la syntaxe des instructions utilisées et d'accorder une attention particulière aux indentations (tabulations).

➤ 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)

Correction :

```
1 from machine import *
2 from thingz import *
3 import utime
4
5 p19 = Pin(13, Pin.IN)
6 nb_ours=0
7
8 while True:
9     capt_optique = p19.value()
10    if capt_optique == 1:
11        nb_ours = nb_ours + 1
12        print('Ours detecte(s)='+str(nb_ours))
13        utime.sleep(1)
```

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.

Correction :

```
1 from machine import *
2 from thingz import *
3 import utime
4
5 p19 = Pin(13, Pin.IN)
6 nb_ours=0
7 t0 = utime.ticks_ms()
8
9 while True:
10     capt_optique = p19.value()
11     if capt_optique == 1:
12         nb_ours = nb_ours + 1
13         print('Ours detecte(s)='+str(nb_ours))
14         print(str(round(utime.ticks_diff(utime.ticks_ms(), t0)/1e3)))
15         utime.sleep(1)
```

➤ Bilan commun :

Dans le milieu industriel, la programmation s'effectue principalement en langage textuel notamment en Micro Python. Ce terme désigne une version du langage Python spécifiquement optimisée et adaptée aux contraintes des microcontrôleurs. Le code est saisi à l'aide d'instructions textuelles précises et standardisées. Son écriture impose de respecter rigoureusement l'indentation (tabulation). En programmation, une indentation est obligatoirement insérée devant chaque commande située à l'intérieur d'une structure algorithmique (comme une boucle ou une condition) afin d'en délimiter le bloc d'instruction.

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)
```