

# INFEM

## Informatique embarquée

Mikaël Briday, Pierre Molinaro



- 1 Installation/exécution Python 3
- 2 TP Temps Réel : Bac

1 Installation/exécution Python 3

2 TP Temps Réel : Bac

Le TP Temps réel du module INFEM utilise le langage Python. Pour l'utiliser à distance, il sera nécessaire d'installer :

- ») l'environnement *Python 3.x* (pas de Python 2.x) ;
- ») le framework *PyQt* ;
- ») la bibliothèque *fysom*

Cette installation n'est nécessaire que si vous souhaitez installer les outils sur votre ordinateur personnel. Sur les Mac de la salle D102, l'installation est déjà effectuée.

# Installation sous MS Windows

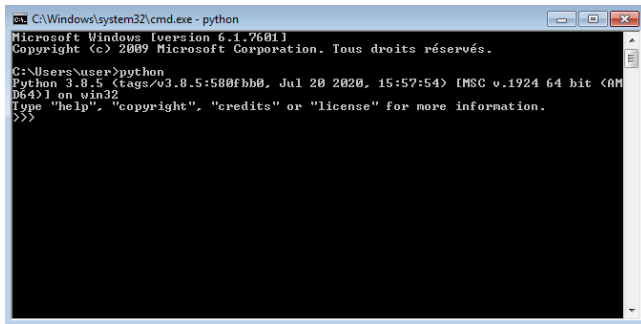
L'environnement Python (3.11.5) peut être téléchargé sur  
<https://www.python.org/downloads/release/python-3115/> :

» « Windows installer (64-bit) » pour une version Windows 64bits

Lors de l'installation, cocher la case *Add Python 3.11 to PATH* puis  
Install Now.

# Installation sous MS Windows

Vous pouvez tester la bonne installation en ouvrant une invite de commande (rechercher l'outil cmd dans le menu Windows) et en tapant python :



```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

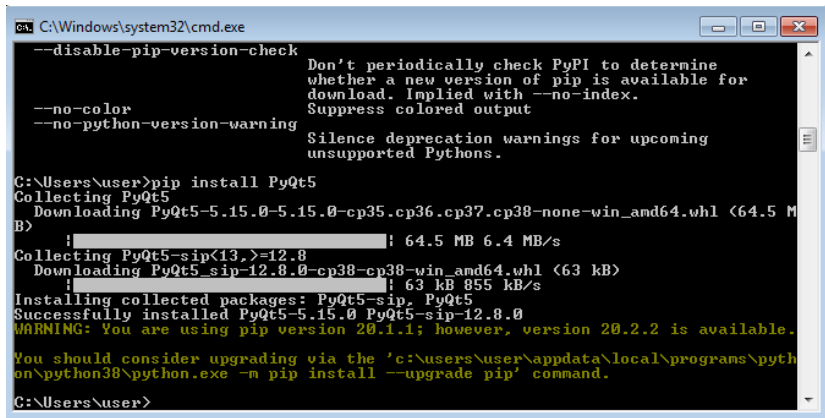
C:\Users\user>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

On peut quitter avec la commande `quit()`.

# Installation sous MS Windows

L'installation de **PyQt5** se fait avec l'outil pip installé avec Python. Dans une fenêtre de commande (cmd.exe) :

```
pip install PyQt5
```

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the command "pip install PyQt5" being executed. The output displays the collection and download of two packages: "PyQt5" (64.5 MB) and "PyQt5-sip" (63 kB). It also shows a warning about the pip version (20.1.1) and a suggestion to upgrade. The installation is successful.

```
C:\Windows\system32\cmd.exe

--disable-pip-version-check    Don't periodically check PyPI to determine
                               whether a new version of pip is available for
                               download. Implied with --no-index.
--no-color                    Suppress colored output
--no-python-version-warning    Silence deprecation warnings for upcoming
                               unsupported Pythons.

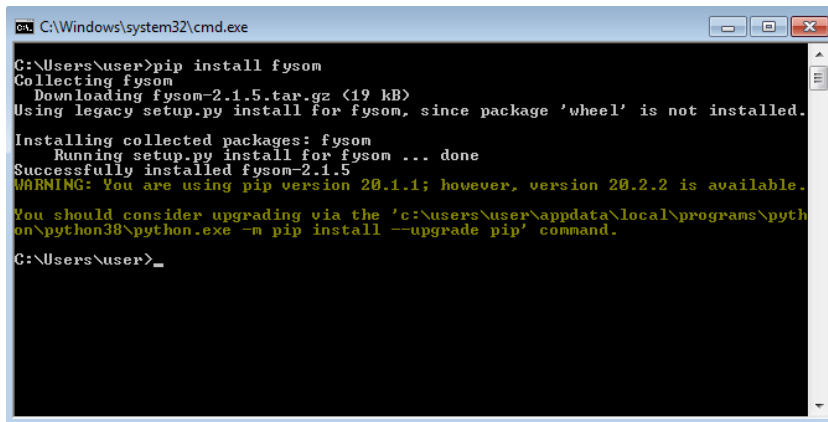
C:\Users\user>pip install PyQt5
Collecting PyQt5
  Downloading PyQt5-5.15.0-5.15.0-cp35.cp36.cp37.cp38-none-win_amd64.whl (64.5 MB)
    |#####| 64.5 MB 6.4 MB/s
Collecting PyQt5-sip<13,>=12.8
  Downloading PyQt5_sip-12.8.0-cp38-cp38-win_amd64.whl (63 kB)
    |#####| 63 kB 855 kB/s
Installing collected packages: PyQt5-sip, PyQt5
Successfully installed PyQt5-5.15.0 PyQt5-sip-12.8.0
WARNING: You are using pip version 20.1.1; however, version 20.2.2 is available.
You should consider upgrading via the 'c:\users\user\appdata\local\programs\python\python38\python.exe -m pip install --upgrade pip' command.

C:\Users\user>
```

# Installation sous MS Windows

De même, la bibliothèque externe Fysom est nécessaire :

```
pip install fysom
```



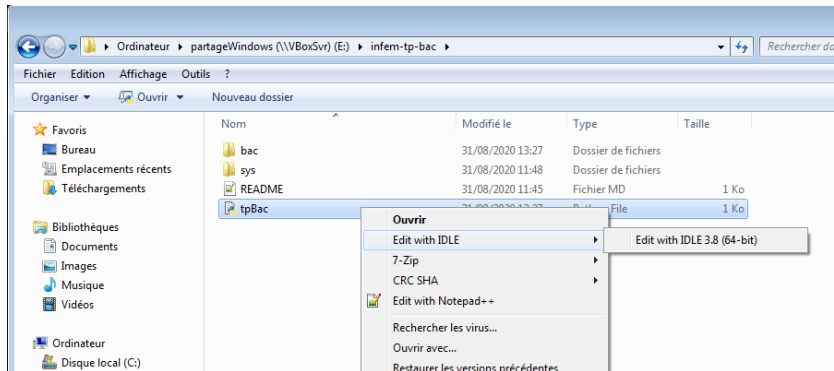
```
C:\Windows\system32\cmd.exe

G:\Users\user>pip install fysom
Collecting fysom
  Downloading fysom-2.1.5.tar.gz (19 kB)
Using legacy setup.py install for fysom, since package 'wheel' is not installed.
Installing collected packages: fysom
  Running setup.py install for fysom ... done
Successfully installed fysom-2.1.5
WARNING: You are using pip version 20.1.1; however, version 20.2.2 is available.
You should consider upgrading via the 'c:\users\user\appdata\local\programs\python\python38\python.exe -m pip install --upgrade pip' command.
G:\Users\user>_
```



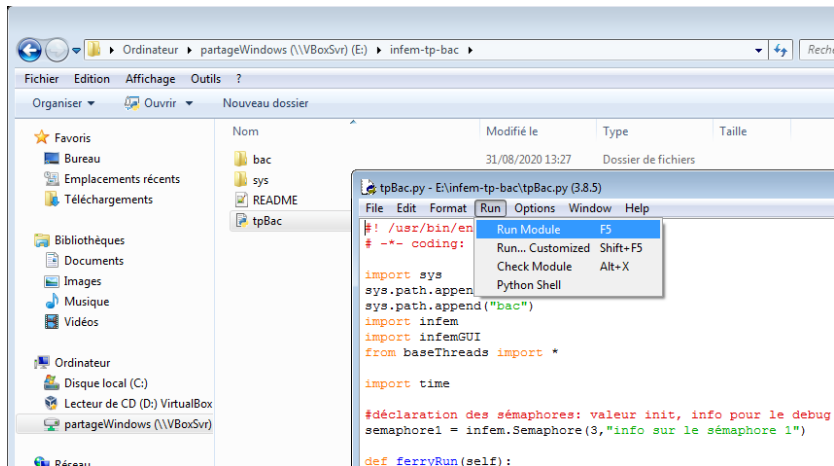
# Utilisation sous MS Windows

Le fichier `tpBac.py` est à modifier. Vous pouvez l'ouvrir avec l'éditeur intégré à Python :



# Utilisation sous MS Windows

Le lancement de l'application se fait avec un double-clic sur `tpBac.py` dans l'explorateur, ou avec la touche F5 dans l'environnement IDLE.



Sous Linux, il faut s'assurer d'avoir une version à jour de Python 3. Sous Debian (ou Ubuntu) :

```
sudo apt install python3
```

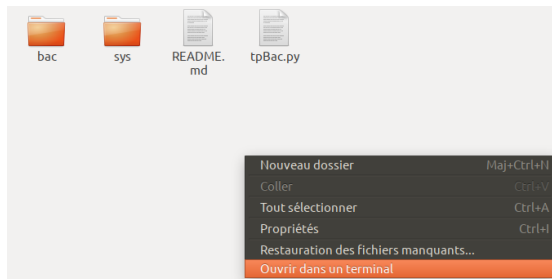
Puis installer PyQt5 avec le gestionnaire de paquet ou pip, et finalement fysom avec pip :

```
pip3 install pyqt5 fysom
```

Note : on utilise ici pip3 pour enlever l'ambiguïté quand il y a co-existence entre Python 2 et 3...

# Utilisation sous GNU/Linux

Le fichier à modifier est `tpBac.py`, à ouvrir avec un éditeur de texte.  
Pour lancer l'application, il faut utiliser le terminal (démarrage du terminal ici à partir de l'explorateur sous Gnome) :



puis de lancer le script comme une application :

```
./tpBac.py
```

Sous Mac OS X, jusqu'à la version 10.15 (Catalina), la version de Python par défaut est Python 2.x. Il est nécessaire d'installer une version plus récente :

- » soit directement sur le site  
<https://www.python.org/downloads/release/python-3115/>
- » soit à travers un système de paquet si vous en utilisez déjà un (MacPort, HomeBrew,...)

Sur le site, vous récupérez directement un paquet d'installation (.pkg)

Puis installer PyQt5 et fysom l'outil pip avec la commande suivante dans le terminal (Application ⇒ Utilitaires ⇒ Terminal) :

```
pip3 install pyqt5 fysom
```

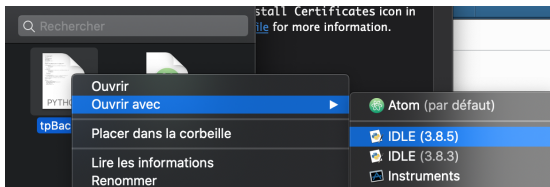
Note : on utilise ici pip3 pour enlever l'ambiguïté quand il y a co-existence entre Python 2 et 3...

Il faut lancer l'installation en tant que super-utilisateur.

# Utilisation sous Mac

Le fichier à modifier est `tpBac.py`.

Pour ouvrir le fichier, vous pouvez faire un clic droit sur le fichier et choisir : ouvrir avec  $\Rightarrow$  IDLE (*version*)



Vous pouvez alors éditer le script et le lancer avec F5 (ou menu Run  $\Rightarrow$  Run module)

## Attention

Sur les MAC de la D102, IDLE ne démarre pas correctement, il faut le lancer avec le terminal

# Utilisation sous Mac en D102

Pour l'utilisation en D102, il faut démarrer un terminal (`Terminal.app` dans Applications/) et démarrer `idle` directement.

On considère ici que le dossier de TP (`infem-tp-bac-master`) a été décompressé sur le bureau (`Desktop/`) :

```
idle3 Desktop/infem-tp-bac/tpBac.py
```

Une fois IDLE démarré, vous n'aurez plus besoin du Terminal.



- 1 Installation/exécution Python 3
- 2 TP Temps Réel : Bac

Le TP utilise le langage Python, associé à l'environnement PyQt5. Il peut être exécuté sous Windows, Linux ou Mac. Un ensemble de primitive est fourni pour :

- » *déclarer* des tâche temps réel ;
- » *démarrer* ces tâches temps réel ;
- » assurer la *synchronisation* avec des sémaphores

Dans le TP, les tâches seront déjà déclarées et démarrées, on s'intéressera uniquement à la synchronisation.

Uniquement le fichier `tpBac.py` sera à modifier.

Vous rendrez le listing commenté du fichier `tpBAC.py` *en PDF* sur Hippocampus.

Le TP comporte 3 étapes, versions A, B et C, chacune s'appuyant sur la première. L'ordre de grandeur du temps à passer est le suivant :

- » la première version A permet de prendre en main l'application. Cette version devrait être réalisée avant la fin de la première séance ;
- » la version B est la plus complexe, elle est généralement terminée durant la première moitié de la deuxième séance ;
- » la dernière version (C) est plus simple si vous avez bien compris la version B. Elle peut être traitée en moins de 2h.

Outre les primitives permettant la création et le démarrage des tâches, vous disposez de fonctions pour la gestion du temps et des synchronisations

# Primitives - gestion du temps

La gestion du temps est définie par le module *time* (doc sur <https://docs.python.org/fr/3/library/time.html>).

Notamment :

```
t1 = time.time() #nb de secondes depuis 1/1/1970 (Unix epoch)
process()
duree = time.time() - t1 #duree contient la durée de process()
et une attente :
time.sleep(1.2) # attend 1.2s
```

Déclaration d'un sémaphore `sem1`, initialisé à 2 :

```
sem1 = infem.Semaphore(2, "synchro_de_...")
```

Les paramètres :

- ») la valeur initiale du sémaphore
- ») une chaîne de caractère d'information (visible dans l'interface graphique)

Un sémaphore est déclaré en dehors de toute fonction (objet global).

## Acquisition du sémaphore

```
sem1.acquire() #acquisition du sémaphore  
sem1.P()       #idem, nom historique
```

Cette primitive teste le sémaphore.

- ») Si sa valeur est strictement positive, il est décrémenté et la fonction retourne immédiatement ;
- ») Si sa valeur négative ou nulle, la primitive est bloquante. La tâche qui invoque cette primitive est mise en attente jusqu'à ce que le sémaphore devienne positif ;

```
sem1.release() #libération du sémaphore  
sem1.V()       #idem, nom historique
```

Cette primitive incrémente le sémaphore. L'appel est non bloquant.



Attente combinée de 2 sémaphores :

`infem.ou2Semaphore(s1,s2)` #où s1 et s2 sont des sémaphores

Cette primitive exécute `s1.P()` ou `s2.P()`, la première qui devient non bloquante.

Si les 2 sémaphores sont non bloquants, c'est le premier qui est prioritaire (`s1.P()` est exécuté).

La valeur de retour est :

- ») 1 si `s1.P()` est exécuté
- ») 2 si `s2.P()` est exécuté

# Primitives - synchronisation

Attente combinée de 2 sémaphores avec échéance :

#s1 et s2 sont des sémaphores, timeout un float  
infem.ou2Semaphore(s1,s2,timeout)

Cette primitive exécute s1.P() ou s2.P(), la première qui est ou devient non bloquante, *à condition que celà se produise avant l'échéance.*

L'échéance (timeout) est une durée (type **float**)

La valeur de retour est :

- » 1 si s1.P() est exécuté
- » 2 si s2.P() est exécuté
- » 3 si l'échéance est atteinte sans que s1.P() ou s2.P() soit exécuté.

## Note

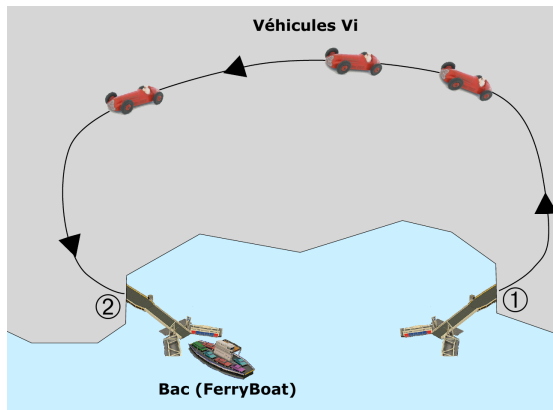
le timeout ne doit pas être nul pour 'tester' un sémaphore !

On désire simuler le comportement d'un ensemble de véhicules qui effectuent un parcours consistant à recommencer le même trajet indéfiniment :

- » effectuer un parcours sur terre ;
- » embarquer à bord d'un bac ;
- » en débarquer lorsque le bac est arrivé à destination.

# Description de l'application

Ainsi, l'application générale fonctionne comme si les véhicules  $V_i$  suivaient un circuit fléché comme celui représenté ci-dessous, partiellement sur terre, partiellement sur mer.



Les vitesses des véhicules sont différentes et quelconques.  
Leur nombre est égal à 10 (numérotées de 0 à 9).

# Description de l'application

Le bac (ferry-boat) qui effectue la traversée ne peut embarquer qu'au plus **4** véhicules à la fois, *de manière séquentielle*.

Lorsque l'embarquement est terminé, il se rend en 1, y débarque les véhicules, et retourne à vide en 2.

## Note

Sur l'animation du TP, le point 1 est situé son bord gauche, et le parcours sur terre des véhicules est visualisé par leurs déplacements horizontaux de la gauche vers la droite jusqu'à l'embarcadère.

`self.avancer()` provoque le démarrage du véhicule  $V_i$  du point 1 vers le point 2. La procédure s'arrête dès que la voiture est arrivée au point 2.

`self.embarquer()` simule l'embarquement du véhicule  $V_i$  sur le bac. Le bac doit être à quai pour que le véhicule ne tombe pas à l'eau !

`self.debarquer()` simule le débarquement du véhicule  $V_i$  hors du bac au point 1. Le bac doit être à quai pour que le véhicule ne tombe pas à l'eau !

`self.log('txt')` permet d'afficher txt sur les logs à gauche de la fenêtre

- `self.traverser()` provoque la traversée du bac, du point 2 au point 1.  
La procédure s'arrête dès que le voiture est arrivée au point 1.
- `self.revenir()` provoque la traversée du bac, du point 1 au point 2. La procédure s'arrête dès que le voiture est arrivée au point 2.
- `self.log('txt')` permet d'afficher txt sur les logs à gauche de la fenêtre

# Travail à réaliser

Initialement, chaque voiture est en 1, et le bac est en 2.

On demande d'écrire la simulation de cette application sous forme de tâches :

- » le comportement de chaque véhicule  $V_i$  sera piloté par une tâche. Toutes ces tâches devront être identiques (à l'indice près). Leurs vitesses seront fixées par les paramètres d'une fonction qui simule leur déplacement (fonction fournie).
- » le comportement du bac sera également piloté par une tâche.

## Restriction

- » le mot clé python **break** est interdit.
- » il ne doit plus y avoir d'attente dans votre code `time.sleep()`
- » on ne doit pas utiliser le `ou2Semaphore()` pour essayer de déduire la valeur du sémaphore (en utilisant un timeout à 0)



Le bac ne doit appareiller du point 2 que lorsqu'il est complètement chargé, c'est-à-dire avec 4 véhicules à bord. Votre solution devra réaliser :

- ») l'embarquement *successif* de plusieurs véhicules ;
- ») le débarquement *parallèle* des 4 véhicules.

Dans ce premier cas, on ne se préoccupe pas de la terminaison des tâches. Votre solution devra exprimer la synchronisation au moyen de sémaphores, *sans utiliser de variables globales*.

Note : Il y a quelques informations sur l'utilisation de Python dans la section suivante.

Le bac ne doit appareiller du point 2 que lorsqu'il est complètement chargé, c'est-à-dire avec 4 véhicules à bord. Votre solution devra réaliser :

- ») l'embarquement *successif* de plusieurs véhicules ;
- ») le débarquement *parallèle* des 4 véhicules ;
- ») quand vous cliquez sur le bouton « Demander la terminaison propre », toutes les tâches devront se terminer, c'est-à-dire qu'on doit sortir de la fonction principale de chaque tâche.
- ») l'arrêt doit être au plus tôt : les voitures ne doivent plus rentrer dans le bac ou sortir du bac (en fonction de leur position).

### Note

- » lors de la terminaison, la procédure **def** `terminateCalled(self)` est appelée (`self` ne sera pas utilisé)<sup>a</sup>.
- » si la terminaison de chaque tâche (voiture/bac) est correcte, un **ok!** va apparaître sous le bouton<sup>b</sup>

---

a. Cet appel est fait par la tâche chargée de l'affichage, il est interdit de faire un appel bloquant (ex : `sem.P()`)!

b. vous n'avez pas à programmer ce `Ok!`

Cette version est identique à la version B, si ce n'est que le bac doit appareiller au plus tard 2 secondes après son arrivée au point 2, même s'il ne contient pas 4 véhicules à ce moment-là.

## Note

- » si une voiture est en cours d'embarquement, celui-ci se termine.
- » si une voiture est en attente d'embarquement alors que l'échéance est passée, elle reste à quai.

# Mise au point

Pour la mise au point, vous pouvez afficher du texte avec la fonction **print**, et connaître le nom de la tâche courante est dans le champ `self.name` :

```
print(self.name+":_=>_attente_bac)
```

Attention, Python est un langage fortement typé, et le type des données dans un **print** doit être une chaîne de caractères. Pour permettre l'affichage d'une variable, il faut utiliser **str()** :

```
print(self.name+":_valeur_de_la_variable_loop:_" + str(loop))
```

À la place de la fonction **print()**, vous pouvez utiliser la fonction `self.log()` qui affichera le texte avec les informations générales sur la partie gauche de la fenêtre.

# Rappel rapide - Python

Pour déclarer une variable dans une tâche, il n'y a pas besoin de donner le type :

```
boucle = True #déclaration d'un booléen (True/False)
a = 12.5      #déclaration d'un 'float'
```

Un bloc est défini par son indentation (pas de {} comme en C) :

boucle :	conditions :
<pre>while boucle: #boucle booléen     #code indenté</pre>	<pre>if cond1: #cond1 booléen     #code indenté (interprété si cond1 est vrai) elif cond2: #     #code indenté     #interprété si cond1 est faux, et cond2 vrai else:     #code indenté     #interprété si cond1 et cond2 sont faux</pre>
<pre>for i in range(4):     #code indenté     #4 tours, i de 0 à 3</pre>	