classe Pool

Pool est une classe que permet de paralléliser facilement.

p = Pool(processes=4)
est un ensemble de 4 processus sur lequel une fonction peut être appliquée.

p.map permet de "mapper" la fonction

```
import multiprocessing as mp
def cube(x):
        return x**3
p = mp.Pool(processes = 4)
results = p.map( cube , range(1,7) )
print(results)
```

```
$ python3.4 ExoPool.py
[1, 8, 27, 64, 125, 216]
$
```

Communication entre deux processus par file de messages : Queue

```
import multiprocessing as mp
import sys , time
class maClasse( ) :
        def __init__(self, nom):
                self.nom = nom
        def travailler(self):
                nomProcessus = mp.current_process().name
                print("Travail fait par %s - %s!" % (nomProcessus, self.nom))
def unProcessus(q):
        objet = q.get()
        objet.travailler()
queue = mp.Queue()
p = mp.Process(target=unProcessus, args=(queue,))
p.start()
queue.put(maClasse('un 4IRC'))
queue.close()
p.join()
```

```
$ python3.4 Exo1Queue.py
Travail fait par Process-1 - un 4IRC!
$
```

Communication entre un producteur et un consommateur par file de messages : Queue

```
fin = -1
def producteur(donnees, q):
        print("Production des données et dépôt dans la file q")
        for val in donnees:
                q.put(val)
def consommateur(q):
        while True:
                v = q.get()
                print("Donnée lue = " , v)
                resultat = v * 2
                print("Résultat = " , resultat)
                if v is fin:
                        break
q = mp.Queue()
valeurs = [5, 10, 13, -1]
processProd = mp.Process(target=producteur, args=(valeurs, q))
processConso = mp.Process(target=consommateur, args=(q,))
processProd.start() ; processConso.start()
q.close()
processProd.join() ; processConso.join()
```

La classe Queue :
Créer un système avec double queue :
Une queue utilisée pour déposer des valeurs initiales.
On supposera que chaque valeur initiale est un multiple de 100000
Une queue utilisée pour récupérer les résultats.
Chaque processus ajoutera son PID à cette valeur.

```
import multiprocessing as mp
def add_pid(queue_in , queue_out) :
      num = mp.current_process().pid
         val = queue_in.get()
         queue_out.put(num + val)

queue0 = mp.Queue()
queue1 = mp.Queue()
[ queue0.put(k*10000) for k in range(4) ]
processes = [ mp.Process(target = add_pid, args=(queue0,queue1,)) for x in range(4)  ]
for p in processes :
      p.start()
for p in processes :
      p.join()
results = [ queue1.get() for p in processes ]
print(results)
```

Communication entre un producteur et plusieurs consommateurs par file de messages : Queue

```
#Utilisation de multiprocessing.Queue
def maFonction(q):
        for i in range(6) :
                v = q.get()
                if v == "fin":
                        break
        print (mp.current_process().name , "a traité %i valeurs " %i)

VALEURS = mp.Queue()
for val in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, "fin", "fin", "fin"]:
        VALEURS.put(val)

lesProcessus = [mp.Process(target=maFonction, args=(VALEURS,)) for i in range(3)]

for p in lesProcessus :
        p.start()
for p in lesProcessus :
        p.join()

print ("Le nombre de valeurs après la fin des processus : " ,VALEURS.qsize())
```

Synchronisation par verrous

```
#Utilisation des verrous
def process1(verrou, stream):
    with verrou:
        time.sleep(2)
        stream.write("Jeton pris par l'instruction with\n")

def process2(verrou, stream):
    verrou.acquire()
    try:
        stream.write("Jeton pris directement\n")
    finally:
        verrou.release()

verrou = mp.Lock()
p1 = mp.Process(target=process1, args=(verrou, sys.stdout))
p2 = mp.Process(target=process2, args=(verrou, sys.stdout))

p1.start()
p2.start()
p1.join()
p2.join()
```

```
$ python3.4 Exo4.py
Jeton pris par l'instruction with
Jeton pris directement
$
```

```
def doubler(nombre):
        resultat = nombre * 2
        pid = os.getpid()
        print("{0} doublé en {1} par le processus : {2}".format(nombre, resultat, pid))

nombres = [5, 10, 15, 20, 25]
procesus = [ ]
for n in nombres :
        p = mp.Process(target=doubler, args=(n,))
        procesus.append(p)
        p.start()
for p in procesus:
        p.join()
```

```
$ python3.4 Exo5.py
5 doublé en 10 par le processus : 7113
10 doublé en 20 par le processus : 7114
20 doublé en 40 par le processus : 7116
15 doublé en 30 par le processus : 7115
25 doublé en 50 par le processus : 7117
$
```

```
def doubler(nombre):
        print("Processus %d – traitement de %d" %( os.getpid() , nombre))
        return nombre * 2
nombres = [5, 10, 20, 4 , 8, 15]
pool = mp.Pool(processes=3)
r = pool.map(doubler, nombres)
print ("RESULTAT = " , r)
```

```
$ python3.4 Exo6.py
Processus 15033 – traitement de 5
Processus 15035 – traitement de 5
Processus 15034 – traitement de 5
Processus 15033 – traitement de 5
Processus 15035 – traitement de 5
Processus 15033 – traitement de 5
RESULTAT = [10, 20, 40, 8, 16, 30]
$
```

Les sémaphores

Création d'un sémaphore

sem = multiprocessing.Semaphore(n)

crée un sémaphore initialisé à n

sem.acquire()

L'opération P(sem)

sem.release()

L'opération V(sem)

Exemple d'utilisation des sémaphores de Dijkstra

```
import multiprocessing as mp
import sys , time, random
def process1(s) :
        print("Je suis le process 1 et j'attends 15 secondes")
        time.sleep(15)
        print("Je suis le process 1 et je génère un jeton [ V(s) ]")
        s.release()
        sys.exit(0)
def process2(s) :
        print("Je suis le process 2 et je me bloque sur le sémaphore : [ P(s) ]")
        s.acquire()
        print("Je suis le process 2 et je suis DEBLOQUE")
        sys.exit(0)

sem = mp.Semaphore(0)          #Création d'un sémaphore
p1 = mp.Process(target = process1 , args = (sem,))
p2 = mp.Process(target = process2 , args = (sem,))
p1.start() ; p2.start()
p1.join() ; p2.join()
sys.exit(0)
```

```
$ python testSemaphore.py
Je suis le process 1 et j'attends 15 secondes
Je suis le process 2 et je me bloque sur le sémaphore : [ P(s) ]
Je suis le process 1 et je génère un jeton [ V(s) ]
Je suis le process 2 et je suis DEBLOQUE
$
```

Exemple d'utilisation des sémaphores de Dijkstra

```
import multiprocessing as mp
import sys , time, random

def fonc(l , i) :

        l.acquire()

        for i in range(20) :

                print ("Bonjour ", i)

        l.release()

lock = mp.Lock()

for num in range(10) :

        p = mp.Process(target = fonc , args = (lock,num) )

        p.start()
```

Mémoire partagée

```
from multiprocessing import Process, Value, Array
def maFonction (n, T) :
    n.value = 3.1415927
    for i in range(len(T)) :
        T[i] = -T[i]

N = Value('d', 0.0)
tab = Array('i', range(10))

p = Process(target=maFonction , args=( N , tab) )
p.start()
p.join()
print(N.value)
print(tab[:])
```

```
3.1415927
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

```
def semaphoreFonc(sema, mutex, val):
        sema.acquire()
        mutex.acquire()
        val.value += 1
        print (running.value, 'taches en execution en parallele')
        mutex.release()
        random.seed()
        time.sleep(random.random()*2)

        mutex.acquire()
        val.value -= 1
        print ("Fin du processus %s" % mp.current_process())
        mutex.release()

        sema.release()
s = mp.Semaphore(3) ; mutex = mp.Lock() ; v = mp.Value('i', 0)
processes = [mp.Process(target=semaphoreFonc, args=(s, mutex, v)) for i in range(10)]
for p in processes:
        p.start()
for p in processes:
        p.join()
```

Exemple d'utilisation des sémaphores de Dijkstra

Exemple d'utilisation des sémaphores de Dijkstra

```
def maFonc(val):
        for i in range(50) :
                time.sleep(0.01)
                val.value += 1


v = mp.Value('i', 0)
processes = [ mp.Process(target=maFonc, args=(v,)) for i in range(10) ]
for p in processes :
                p.start()
for p in processes:
                p.join()
print(v.value)
```

```
$ python test.py
421
$ python test.py
435
$ python test.py
451
$ python test.py
388
$ python test.py
425
$
```