```
pip install simpy
```

```
Collecting simpy
  Downloading https://files.pythonhosted.org/packages/20/f9/874b0bab83406827db
Installing collected packages: simpy
Successfully installed simpy-4.0.1
```

RECURSOS COMPARTIDOS Que los recursos forman un punto de congestión donde los procesos hacen cola para poder usarlos. Simpy define tres categorias de recursos

1. Recursos: Estos pueden ser utilizados por un número limitado de procesos a la vez (Por ejemplo uan estación de servicio con un número limitado de bombas de combustible)
2. Contenedores: Recursos que modelan la producción y el consumo de una bolsa homogéneo. Este puede ser un elemento continuo por ejemplo el agua o discreteo como en número de manzanas
3. Tiendas: Recursos que permite la produccón y el consumo de objetos en python.

Concepto: Los recursos en si es una especie de contenedor con una capacidad generalmente esta capcidad es limitada. Entonces los procesos pueden intentar poner algo en el recurso o sacar algo del mismo. Se el recurso esta lleno o vacio deben hacer cola y esperar. Cada recursos tiene una capacidad máxima y dos colas, una cola para los procesos que quieren poner algo y otra para los procesos que deseas sacar algo. Los metodos son el** put()** y** get()** devuelven un evnto que se activa cuando la acción correspondiente se realiza.

Se puede agregar o llamar a los metodos con el request()/release() que es equivalente al put() y get(). Simpy implementa tres tipos de recursos:

1. Resource
2. PriorityResource: donde los procesos de cola se ordenan por prioridad.
3. PreemptiveResource: Donde los procesos pueden adelantarse a otros procesos con una prioridad mas baja.

```
import simpy
def recurso_usuario(env, recurso):
  print('Utilizo el recurso usuario')
  with recurso.request() as peticion: # Genero un evento de requerimiento
    yield peticion
    imprimir_datos_recurso(recurso)
    yield env.timeout(1)
    recurso.release(peticion) #LIbero el recurso despues de su uso.
    print('Se libera el recurso usuario')


def imprimir_datos_recurso(rec):
  print(f'{rec.count} of {rec.capacity}')
  print(f'Usuarios: {rec.users}')
  print(f' Evento de colas: {rec.queue}')
```

```
env = simpy.Environment()
res = simpy.Resource(env, capacity=1)
#usuario = env.process(recurso_usuario(env, res))
proceso = [env.process(recurso_usuario(env, res)), env.process(recurso_usuario(env,
env.run()
```

```
Utilizo el recurso usuario
Utilizo el recurso usuario
1 of 1
Usuarios: [<Request() object at 0x7ff174432d50>]
 Evento de colas: [<Request() object at 0x7ff174432950>]
Se libera el recurso usuario
1 of 1
Usuarios: [<Request() object at 0x7ff174432950>]
 Evento de colas: []
Se libera el recurso usuario
```

Recurso con prioridad. Como sabrá del mundo real, no todos son igualmente importantes. Para asignar eso a SimPy, está el PriorityResource . Esta subclase de recurso permite que los procesos de solicitud proporcionen una prioridad para cada solicitud. Las solicitudes más importantes obtendrán acceso al recurso antes que las menos importantes. La prioridad se expresa mediante números enteros; números más pequeños significan una prioridad más alta. Aparte de eso, funciona como un recurso normal .

```
def recurso_usuario(name, env, resource, wait, prio):
  yield env.timeout(wait)
  with resource.request(priority=prio) as req:
    print(f'{name} requerido en tiempo {env.now} con prioridad={prio}')
    yield req
    print(f'{name} obtengo el recurso en el tiempo {env.now}')
    yield env.timeout(3)
    print('Libero el recurso despues de 3 unidades de tiempo')

env = simpy.Environment()
res = simpy.PriorityResource(env, capacity=1)
p1 = env.process(recurso_usuario(1, env, res, wait=0, prio=0))
p2 = env.process(recurso_usuario(2, env, res, wait=1, prio=0))
p3 = env.process(recurso_usuario(3, env, res, wait=2, prio=-1))
env.run()
# Aunque p3 solicitó el recurso más tarde que p2 , podría usarlo antes porque su pr
```

```
1 requerido en tiempo 0 con prioridad=0
1 obtengo el recurso en el tiempo 0
2 requerido en tiempo 1 con prioridad=0
3 requerido en tiempo 2 con prioridad=-1
Libero el recurso despues de 3 unidades de tiempo
3 obtengo el recurso en el tiempo 3
Libero el recurso despues de 3 unidades de tiempo
2 obtengo el recurso en el tiempo 6
Libero el recurso despues de 3 unidades de tiempo
```

Contenedores Los contenedores le ayudan a modelar la producción y el consumo de una bolsa homogéneo e indiferenciado. Puede ser continuo (como el agua) o discreto (como las manzanas).

Puede usar esto, por ejemplo, para modelar el tanque de gas / gasolina de una estación de servicio. Los camiones cisterna aumentan la cantidad de gasolina en el tanque mientras que los automóviles la disminuyen.

El siguiente ejemplo es un modelo muy simple de una estación de servicio con un número limitado de surtidores de combustible (modelado como Resource) y un tanque modelado como Container.

Los contenedores le permiten recuperar tanto su corriente level como su capacity. También puede acceder a la lista de eventos en espera a través de los atributos put queue y get queue

```
class EstacionGasolina():
  def __init__(self, env):
    self.env = env
    self.dispensador = simpy.Resource(env, capacity=2)
    self.tanque = simpy.Container(env, init=40, capacity=1000)
    self.monitoreo = env.process(self.monitoreo_tanque())
  def monitoreo_tanque(self):
    while True:
      if self.tanque.level < 100:
        print(f'Llamar al tanquero {self.env.now}')
        env.process(tanquero(self.env, self))
      yield env.timeout(15)

def tanquero(env, estacion):
  yield env.timeout(10)
  print(f'Llega el tanquero en el tiempo {env.now}')
  nivel = estacion.tanque.capacity - estacion.tanque.level # Valor que necesita par
  yield estacion.tanque.put(nivel)

def carro(nombre, env, estacion):
  print(f'El carro {nombre} llega al tiempo {env.now}')
  with estacion.dispensador.request() as requerimiento:
    yield requerimiento
    print(f'El carro {nombre} se esta llenando {env.now}')
    yield estacion.tanque.get(40)
    yield env.timeout(5)
    print(f'El carro {nombre} acabo de llenar su tanque en el tiempo {env.now}')

def generador_carros(env, estacion):
  for i in range(4):
    env.process(carro(i, env, estacion))
    yield env.timeout(5)

env = simpy.Environment()
estacion = EstacionGasolina(env)
generado = env.process(generador_carros(env, estacion))
env.run(until=40)
```

```
Llamar al tanquero 0
El carro 0 llega al tiempo 0
El carro 0 se esta llenando 0
El carro 1 llega al tiempo 5
El carro 0 acabo de llenar su tanque en el tiempo 5
El carro 1 se esta llenando 5
Llega el tanquero en el tiempo 10
El carro 2 llega al tiempo 10
El carro 2 se esta llenando 10
El carro 3 llega al tiempo 15
El carro 1 acabo de llenar su tanque en el tiempo 15
El carro 2 acabo de llenar su tanque en el tiempo 15
El carro 3 se esta llenando 15
El carro 3 acabo de llenar su tanque en el tiempo 20
```

Tiendas Usando Tiendas puede modelar la producción y el consumo de objetos concretos (en contraste con la “cantidad” bastante abstracta almacenada en contenedores). Una sola tienda puede incluso contener varios tipos de objetos.

Además Store, hay una FilterStore que te permite usar una función personalizada para filtrar los objetos que sacas de la tienda y PriorityStore dónde salen los artículos de la tienda en orden de prioridad.

Al igual que con los otros tipos de recursos, puede obtener la capacidad de una tienda a través del capacity atributo. El atributo items apunta a la lista de artículos actualmente disponibles en la tienda. Se puede acceder a las colas de colocación y obtención a través de los atributos put_queuey get_queue.

Aquí hay un ejemplo simple que modela un escenario genérico de productor / consumidor:

```
def productor(env, tienda):
  for i in range(100):
    yield env.timeout(2)
    yield tienda.put(f'pieza {i}')
    print(f'Se genero pieza {i} piezas en el tiempo {env.now}')

def consumidor(nombre, env, tienda):
  while True:
    yield env.timeout(1)
    print(f'{nombre} pide consumir en el timepo {env.now}')
    item = yield tienda.get()
    print(f'{nombre} obtubo el item {item} en el tiempo {env.now}')
env = simpy.Environment()
tienda = simpy.Store(env, capacity=1)

productor = env.process(productor(env, tienda))
consumidor = env.process(consumidor("1", env, tienda))
env.run(until=10)
```

```
1 pide consumir en el timepo 1
Se genero pieza 0 piezas en el tiempo 2
1 obtubo el item pieza 0 en el tiempo 2
1 pide consumir en el timepo 3
Se genero pieza 1 piezas en el tiempo 4
```

```
1 obtubo el item pieza 1 en el tiempo 4
1 pide consumir en el timepo 5
Se genero pieza 2 piezas en el tiempo 6
1 obtubo el item pieza 2 en el tiempo 6
1 pide consumir en el timepo 7
Se genero pieza 3 piezas en el tiempo 8
1 obtubo el item pieza 3 en el tiempo 8
1 pide consumir en el timepo 9
```

Con a PriorityStore, podemos modelar elementos de diferentes prioridades. En el siguiente ejemplo, un proceso de inspector encuentra y registra problemas que un proceso de mantenimiento independiente repara en orden de prioridad.

```
def inspector(env, problemas):
  for problema in  [simpy.PriorityItem('P2', '#0000'), simpy.PriorityItem('P0', '#0
    yield env.timeout(1)
    print(f'En el tiempo {env.now} se genero el problema {problema}')
    yield problemas.put(problema)

def mantenimiento(env, problemas):
  while True:
    problema = yield problemas.get()
    yield env.timeout(3)
    print(f'El problema {problema} esta reparado en el tiempo {env.now}')

env = simpy.Environment()
problemas = simpy.PriorityStore(env)
env.process(inspector(env, problemas))
env.process(mantenimiento(env, problemas))
env.run(until=20)
```

```
En el tiempo 1 se genero el problema PriorityItem(priority='P2', item='#0000')
En el tiempo 2 se genero el problema PriorityItem(priority='P0', item='#0001')
En el tiempo 3 se genero el problema PriorityItem(priority='P3', item='#0002')
El problema PriorityItem(priority='P2', item='#0000') esta reparado en el tiem
En el tiempo 4 se genero el problema PriorityItem(priority='P1', item='#0003')
El problema PriorityItem(priority='P0', item='#0001') esta reparado en el tiem
El problema PriorityItem(priority='P1', item='#0003') esta reparado en el tiem
El problema PriorityItem(priority='P3', item='#0002') esta reparado en el tiem
```

Ejercicio/Tarea

Utilizando las tarea de la predicción de llegadas de vacunas y el recinto de vacunación, realizar un sistema que permita simular y correlacionar el procesos de llegada/compras de vacuna con el procesos de vacunación, en donde si no se tiene un stock/número de vacunas las personas tendran que esperar/reasignar a otro día en donde exista vacunas dentro del establecimiento y realizar el proceso de vacunación.