

In [2]: `1 pip install simpy`

```
Collecting simpy
  Downloading https://files.pythonhosted.org/packages/20/f9/874b0b
ab83406827db93292a5bbe5acb5c18e3cea665b2f6e053292cb687/simpy-4.0.1
-py2.py3-none-any.whl (https://files.pythonhosted.org/packages/20/
f9/874b0bab83406827db93292a5bbe5acb5c18e3cea665b2f6e053292cb687/si
mpy-4.0.1-py2.py3-none-any.whl)
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 categorías de recursos

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

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

Se puede agregar o llamar a los métodos 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.

```
In [6]: 1 import simpy
2 def recurso_usuario(env, recurso):
3     print('Utilizo el recurso usuario')
4     with recurso.request() as peticion: # Genero un evento de req
5         yield peticion
6         imprimir_datos_recurso(recurso)
7         yield env.timeout(1)
8         recurso.release(peticion) #Libero el recurso despues de su
9         print('Se libera el recurso usuario')
10
11
12 def imprimir_datos_recurso(rec):
13     print(f'{rec.count} of {rec.capacity}')
14     print(f'Usuarios: {rec.users}')
15     print(f'Evento de colas: {rec.queue}')
16
```

```

17 env = simpy.Environment()
18 res = simpy.Resource(env, capacity=1)
19 #usuario = env.process(recurso_usuario(env, res))
20 proceso = [env.process(recurso_usuario(env, res)), env.process(
21 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.

```

In [8]: 1 def recurso_usuario(name, env, resource, wait, prio):
        2     yield env.timeout(wait)
        3     with resource.request(priority=prio) as req:
        4         print(f'{name} requerido en tiempo {env.now} con prioridad=
        5         yield req
        6         print(f'{name} obtengo el recurso en el tiempo {env.now}')
        7         yield env.timeout(3)
        8         print('Libero el recurso despues de 3 unidades de tiempo')
        9
       10 env = simpy.Environment()
       11 res = simpy.PriorityResource(env, capacity=1)
       12 p1 = env.process(recurso_usuario(1, env, res, wait=0, prio=0))
       13 p2 = env.process(recurso_usuario(2, env, res, wait=1, prio=0))
       14 p3 = env.process(recurso_usuario(3, env, res, wait=2, prio=-1))
       15 env.run()
       16 # Aunque p3 solicitó el recurso más tarde que p2 . podría usarlo

```

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

out_queue y det_queue

```
In [11]: 1 class EstacionGasolina():
2         def __init__(self, env):
3             self.env = env
4             self.dispensador = simpy.Resource(env, capacity=2)
5             self.tanque = simpy.Container(env, init=40, capacity=1000)
6             self.monitoreo = env.process(self.monitoreo_tanque())
7         def monitoreo_tanque(self):
8             while True:
9                 if self.tanque.level < 100:
10                    print(f'Llamar al tanquero {self.env.now}')
11                    env.process(tanquero(self.env, self))
12                yield env.timeout(15)
13
14         def tanquero(env, estacion):
15             yield env.timeout(10)
16             print(f'Llega el tanquero en el tiempo {env.now}')
17             nivel = estacion.tanque.capacity - estacion.tanque.level # Va
18             yield estacion.tanque.put(nivel)
19
20         def carro(nombre, env, estacion):
21             print(f'El carro {nombre} llega al tiempo {env.now}')
22             with estacion.dispensador.request() as requerimiento:
23                 yield requerimiento
24                 print(f'El carro {nombre} se esta llenando {env.now}')
25                 yield estacion.tanque.get(40)
26                 yield env.timeout(5)
27                 print(f'El carro {nombre} acabo de llenar su tanque en el t:
28
29         def generador_carros(env, estacion):
30             for i in range(4):
31                 env.process(carro(i, env, estacion))
32             yield env.timeout(5)
33
34         env = simpy.Environment()
35         estacion = EstacionGasolina(env)
36         generado = env.process(generador_carros(env, estacion))
37         env.run(until=40)
38
```

```
Llamar al tanquero 0
El carro 0 llega al tiempo 0
El carro 0 se está llenando 0
```

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_queue y get_queue.

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

```
In [14]: 1 def productor(env, tienda):
2         for i in range(100):
3             yield env.timeout(2)
4             yield tienda.put(f'pieza {i}')
5             print(f'Se genero pieza {i} piezas en el tiempo {env.now}')
6
7         def consumidor(nombre, env, tienda):
8             while True:
9                 yield env.timeout(1)
10                print(f'{nombre} pide consumir en el tiempo {env.now}')
11                item = yield tienda.get()
12                print(f'{nombre} obtuvo el item {item} en el tiempo {env.now}')
13        env = simpy.Environment()
14        tienda = simpy.Store(env, capacity=1)
15
16        productor = env.process(productor(env, tienda))
17        consumidor = env.process(consumidor("1", env, tienda))
18        env.run(until=10)
```

```
1 pide consumir en el tiempo 1
Se genero pieza 0 piezas en el tiempo 2
1 obtuvo el item pieza 0 en el tiempo 2
1 pide consumir en el tiempo 3
Se genero pieza 1 piezas en el tiempo 4
1 obtuvo el item pieza 1 en el tiempo 4
1 pide consumir en el tiempo 5
Se genero pieza 2 piezas en el tiempo 6
1 obtuvo el item pieza 2 en el tiempo 6
1 pide consumir en el tiempo 7
Se genero pieza 3 piezas en el tiempo 8
1 obtuvo el item pieza 3 en el tiempo 8
1 pide consumir en el tiempo 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.

```
In [16]: 1 def inspector(env, problemas):
2         for problema in [simpy.PriorityItem('P2', '#0000'), simpy.Pr
3             yield env.timeout(1)
4             print(f'En el tiempo {env.now} se genero el problema {proble
5             yield problemas.put(problema)
6
7         def mantenimiento(env, problemas):
8             while True:
9                 problema = yield problemas.get()
10                yield env.timeout(3)
11                print(f'El problema {problema} esta reparado en el tiempo {
12
13 env = simpy.Environment()
14 problemas = simpy.PriorityStore(env)
15 env.process(inspector(env, problemas))
16 env.process(mantenimiento(env, problemas))
17 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 tiempo 4

En el tiempo 4 se genero el problema PriorityItem(priority='P1', item='#0003')

El problema PriorityItem(priority='P0', item='#0001') esta reparado en el tiempo 7

El problema PriorityItem(priority='P1', item='#0003') esta reparado en el tiempo 10

El problema PriorityItem(priority='P3', item='#0002') esta reparado en el tiempo 13

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.

```
In [15]: 1 #SE USA CODIGO DE LLEGADA VACUNACION
2 import simpy
3 import random
4 import matplotlib.pyplot as pp
5 import numpy as np
6 %matplotlib inline
7 #PARAMETROS
8
9 HOSPITALES = 2
10
11
12 HOSPITAL_VENT_A = 7
13 HOSPITAL_VENT_B = 4
14
15 HOSPITAL_CAMA_A = 7
```

```

16 HOSPITAL_CAMA_B = 4
17
18 INFECTADOS = 20
19 TASA_CRECIMIENTO = 7
20 PERSONAL_MEDICO = 150
21 DIAS_INTERNADO = 10
22 DIAS_SIMULACION = 20
23
24 CANTIDAD_VACUNAS = 50
25
26
27
28 #Diccionario para almacenar los resultados
29 persona_recuperadas={}
30 persona_fallecidas={}
31
32
33 class Hospital(object):
34     #constructor
35     def __init__(self, env, num_cama, name):
36         self.env = env
37         self.num_cama = num_cama
38         self.camas = simpy.Resource(env, num_cama)
39         self.name = name
40         self.cantidadvacu = 50
41
42
43     def ingresar_paciente(self, paciente):
44         yield self.env.timeout(random.randint(DIAS_INTERNADO-5,
45         print("El paciente se termino de atender: ", paciente,
46
47 def llegada_compra(env,paciente):
48     llegada = env.now
49     estado = random.randint(1,100)
50     cantidadc = random.randint(1,10)
51     print("El cliente con id :",paciente,"en tiempo :",random.r
52     c = CANTIDAD_VACUNAS - cantidadc
53     CANTIDAD_VACUNAS = c
54     yield CANTIDAD_VACUNAS
55
56
57
58 def llegada_paciente(env, hospital, paciente):
59     arrive = env.now
60     estado = random.randint(1,100)
61     if (estado < 60):
62         with hospital.camas.request() as cama:
63             dias_esperando = random.randint(1,5) #Numero de dia
64             requerimiento = yield cama | env.timeout(dias_esper
65             wait = env.now - arrive
66             if cama in requerimiento:
67                 print("Al paciente con id : ", paciente, " se l
68                 #SE COMPRUEBA SI LA CANTIDAD DE VACUNAS ES MENC
69
70                 if CANTIDAD_VACUNAS < 10:
71
72                     #SE AGREGA LOS METODOS DE COMPRA DE PROVEED
73                     print("Se ESPERA AL SIGUIENTE DIA : ",wait)
74                     #SE ESPERA AL SIGUIENTE DIA
75                     yield env.timeout(wait)

```

```

76
77         else:
78             print("SE IMPRIME LA CANTIDAD DE VACUNAS :
79                 llegada_compra(env,paciente)
80
81 #             yield env.process(hospital.ingresar_paciente(
82         else:
83             print("El paciente con id : " , paciente, " en
84                 persona_fallecidas[env.now] = persona_fallecida
85     else :
86         print("El paciente con id: ", paciente, " que esta en e
87
88 def ejecutar(env, tasa_crecimiento, infectados):
89     hospitalA = Hospital(env, HOSPITAL_CAMA_A, "A")
90     hospitalB = Hospital(env, HOSPITAL_CAMA_B, "B")
91     for i in range(infectados):
92         asignar_hospital(env, hospitalA, hospitalB, i)
93     paciente = infectados
94     while True:
95         yield env.timeout(1)
96         for i in range(tasa_crecimiento):
97             paciente += 1
98             asignar_hospital(env, hospitalA, hospitalB, pacient
99
100 def asignar_hospital(env, hospitalA, hospitalB, paciente):
101     #SE APLICA LA SELECCION DE PROVEEDOR ALEATORIO
102     hosp_esc = random.randint(1,2)
103     if (hosp_esc == 1):
104         print("Llegada del paciente con id: ", paciente, " PROV
105             env.process(llegada_paciente(env, hospitalA, paciente))
106     else:
107         print("Llegada del paciente con id: ", paciente, " PROV
108             env.process(llegada_paciente(env, hospitalB, paciente))
109
110 print("SIMULACION DE COMPRA DE VACUNACION")
111 env=simpy.Environment()
112 env.process(ejecutar(env,TASA_CRECIMIENTO, INFECTADOS))
113 env.run(until=DIAS_SIMULACION)
114
115 print("Resultados pacientes :")
116 print("Recuperados: ")
117 print(persona_recuperadas)
118 print("Fallecidos: ")
119 print(persona_fallecidas)
120
121 datos=sorted(persona_recuperadas.items()) # Ordenamos los datos
122 x, y =zip(*datos) # Obtener x(tiempo - clave) y el y(Numero de
123 pp.plot(x,y,linewidth=2,color='red') #Dibujamos las lineas
124 pp.scatter(x,y,color='blue') # Dibujamos los puntos (x,y)
125 pp.title("Dias / Personas Recuperadas")
126 pp.grid(True) #Generamos una cuadrícula
127 pp.show() #Mostramos el grafico
128
129 if (persona_fallecidas):
130     datos=sorted(persona_fallecidas.items()) # Ordenamos los da
131     x, y =zip(*datos) # Obtener x(tiempo - clave) y el y(Numero
132     pp.plot(x,y,linewidth=2,color='red') #Dibujamos las lineas
133     pp.scatter(x,y,color='blue') # Dibujamos los puntos (x,y)
134     pp.title("Personas Fallecidas / dias de hospitalizacion")
135     pp.grid(True) #Generamos una cuadrícula

```

```
136     pp.show() #Mostramos el grafico
137 #OBJETOS
138
139 #PROCESOS
140 COMPRAS=3
141
142 #VALIDACION STOCK
143
SIMULACION DE COMPRA DE VACUNACION
Llegada del paciente con id: 0  PROVEEDOR A  en el tiempo de : 0
Llegada del paciente con id: 1  PROVEEDOR B  en el tiempo de : 0
Llegada del paciente con id: 2  PROVEEDOR B  en el tiempo de : 0
Llegada del paciente con id: 3  PROVEEDOR A  en el tiempo de : 0
Llegada del paciente con id: 4  PROVEEDOR A  en el tiempo de : 0
Llegada del paciente con id: 5  PROVEEDOR B  en el tiempo de : 0
Llegada del paciente con id: 6  PROVEEDOR B  en el tiempo de : 0
Llegada del paciente con id: 7  PROVEEDOR B  en el tiempo de : 0
Llegada del paciente con id: 8  PROVEEDOR B  en el tiempo de : 0
Llegada del paciente con id: 9  PROVEEDOR A  en el tiempo de : 0
Llegada del paciente con id: 10 PROVEEDOR A  en el tiempo de : 0
Llegada del paciente con id: 11 PROVEEDOR B  en el tiempo de : 0
Llegada del paciente con id: 12 PROVEEDOR A  en el tiempo de : 0
Llegada del paciente con id: 13 PROVEEDOR B  en el tiempo de : 0
Llegada del paciente con id: 14 PROVEEDOR A  en el tiempo de : 0
Llegada del paciente con id: 15 PROVEEDOR A  en el tiempo de : 0
Llegada del paciente con id: 16 PROVEEDOR B  en el tiempo de : 0
Llegada del paciente con id: 17 PROVEEDOR A  en el tiempo de : 0
Llegada del paciente con id: 18 PROVEEDOR B  en el tiempo de : 0
```

In []: 1