Getting started with simpy

```
!pip install simpy
```

```
Collecting simpy
  Downloading https://files.pythonhosted.org/packages/20/f9/874b0bab83406827db93292a5bbe
Installing collected packages: simpy
Successfully installed simpy-4.0.1
```

Example

```
def car(env):
  while True:
    print("strat parking at %d",env.now)
    parking_duration=5
    yield env.timeout(parking_duration)
    print("strat driving at %d",env.now)
    trip_duration=4
    yield env.timeout(trip_duration)
def bike(env):
  while True:
    print("strat parking at %d",env.now)
    parking_duration=5
    yield env.timeout(parking_duration)
    print("strat driving at %d",env.now)
    trip_duration=4
    yield env.timeout(trip_duration)
```

```
import simpy
env=simpy.Environment()
env.process(car(env))
env.run (until=20)

env.process(bike(env))
env.run (until=30)
```

```
strat parking at %d 0
strat driving at %d 5
strat parking at %d 9
strat driving at %d 14
strat parking at %d 18
strat parking at %d 20
strat driving at %d 23
strat driving at %d 25
strat parking at %d 27
strat parking at %d 29
```

```
! pip install simpy
```

```
Collecting simpy
  Downloading https://files.pythonhosted.org/packages/20/f9/874b0bab83406827db93292a5bbe
Installing collected packages: simpy
Successfully installed simpy-4.0.1
```

Lab 2:

Process interaction:

```
class Car(object):
  def __init__(self, env):
   self.env = env
  # Start the run process everytime an instance is created. ...
   self.action = env.process(self.run())
  def run(self):
    while True:
      print('Start parking and charging at %d' % self.env.now)
      charge_duration = 5
      # We yield the process that process() returns
      # to wait for it to finish
      yield self.env.process(self.charge(charge_duration))
      # The charge process has finished and
      # we can start driving again.
      print('Start driving at %d' % self.env.now)
      trip_duration = 2
      yield self.env.timeout(trip_duration)
  def charge(self, duration):
   yield self.env.timeout(duration)


import simpy
env = simpy.Environment()
car = Car(env)
env.run(until=15)
```

```
Start parking and charging at 0
Start driving at 5
Start parking and charging at 7
Start driving at 12
Start parking and charging at 14
```

```
env.run()
```

Shared resources example:

```
def car(env, name, bcs, driving_time, charge_duration):
  yield env.timeout(driving_time)
  print('%s arriving at %d' % (name, env.now))
  with bcs.request() as req:
    yield req
    print('%s starting to charge at %s' % (name, env.now))
    yield env.timeout(charge_duration)
    print('%s leaving the bcs at %s' % (name, env.now))


import simpy
env = simpy.Environment()
bcs = simpy.Resource(env, capacity=2)



for i in range(4):
  env.process(car(env, 'Car %d' % i, bcs, i*2, 5))


env.run()
```

```
Car 0 arriving at 0
Car 0 starting to charge at 0
Car 1 arriving at 2
Car 1 starting to charge at 2
Car 2 arriving at 4
Car 0 leaving the bcs at 5
Car 2 starting to charge at 5
Car 3 arriving at 6
Car 1 leaving the bcs at 7
Car 3 starting to charge at 7
Car 2 leaving the bcs at 10
Car 3 leaving the bcs at 12
```

Interupting another process:

```
def driver(env, car):
  yield env.timeout(3)
  car.action.interrupt()
class Car(object):
  def __init__(self, env):
    self.env = env
    self.action = env.process(self.run())
  def run(self):
    while True:
      print('Start parking and charging at %d' % self.env.now)
      charge_duration = 5
      # We may get interrupted while charging the battery
      try:
        yield self.env.process(self.charge(charge_duration))
      except simpy.Interrupt:
```

```
        except simpy.Interrupt:
        # When we received an interrupt, we stop charging and... # switch to the "driving" stat
          print('Was interrupted. Hope, the battery is full enough ...')
          print('Start driving at %d' % self.env.now)
          trip_duration = 2
          yield self.env.timeout(trip_duration)

    def charge(self, duration):
      yield self.env.timeout(duration)


env = simpy.Environment()
car = Car(env)
env.process(driver(env, car))
env.run(until=15)
```

```
     Start parking and charging at 0
     Was interrupted. Hope, the battery is full enough ...
     Start driving at 3
     Start parking and charging at 5
     Start parking and charging at 10
```

```
! pip install simpy
```

```
     Collecting simpy
       Downloading https://files.pythonhosted.org/packages/20/f9/874b0bab83406827db93292a5bbe
     Installing collected packages: simpy
     Successfully installed simpy-4.0.1
```

Lab Experiment ->3: (Simpy Simple Simulation)

Q1. Simulate a registration process for registration of students at a counter for the person sitting at the counter. Assume registration takes 5 units of time and other 3 units of time he/she has to wait for other student to arrive.

```
! pip install simpy
```

```
     Collecting simpy
       Downloading https://files.pythonhosted.org/packages/20/f9/874b0bab83406827db93292a5bbe
     Installing collected packages: simpy
     Successfully installed simpy-4.0.1
```

```
def registration(env):
    while True:
        print('Starting Registration process of student: ' , env.now)
        regis_duration = 5
        print('Registration process of student Finished: ' , env.now+5)
        yield env.timeout(regis_duration)
```

```
        print('Waiting for other Student: ' , env.now)
        wait_duration = 3
        print('Student Arrived at Counter: ' , env.now+3)
        yield env.timeout(wait_duration)

import simpy
env = simpy.Environment()
env.process(registration(env))
env.run(until=13)
```

```
 Starting Registration process of student:  0
 Registration process of student Finished:  5
 Waiting for other Student:  5
 Student Arrived at Counter:  8
 Starting Registration process of student:  8
 Registration process of student Finished:  13
```

Q2. In the above simulation, the person performing registration must go to meet his officer for approvals after every 10 units of time. Simulate the environment for that scenario also.

```
import simpy

class registration_process(object):
    def __init__(self, env):
        self.env = env
        self.action = env.process(self.run())

    def run(self):
        while True:
            print('Starting Registration process of student: ' , env.now)
            t1=env.now
            regis_duration = 5
            try:
                yield self.env.process(self.reg_process(regis_duration))
                print('Registration process of student Finished: ' , env.now)
            except simpy.Interrupt:
                print('Going to meet the officer for approvals: ' , env.now)
                print('Student at counter waiting for person to return: '
                      , env.now)
                yield self.env.timeout(1)
                print('Person came back to counter: ' , env.now)
                t2=env.now
                yield self.env.process(self.reg_process(t2-t1))
                print('Registration process of student Finished: ' , env.now)

            print('Waiting for other Student: ' , env.now)
            wait_duration = 3
            print('Student Arrived at Counter: ' , env.now+3)
```

```
            yield env.timeout(wait_duration)

    def reg_process(self, duration):
        yield self.env.timeout(duration)

def officer(env, registration_process):
    while True:
        yield env.timeout(10)
        registration_process.action.interrupt()

env = simpy.Environment()
reg = registration_process(env)
env.process(officer(env, reg))
env.run(until=50)
```

```
 Starting Registration process of student:  0
 Registration process of student Finished:  5
 Waiting for other Student:  5
 Student Arrived at Counter:  8
 Starting Registration process of student:  8
 Going to meet the officer for approvals:  10
 Student at counter waiting for person to return:  10
 Person came back to counter:  11
 Registration process of student Finished:  14
 Waiting for other Student:  14
 Student Arrived at Counter:  17
 Starting Registration process of student:  17
 Going to meet the officer for approvals:  20
 Student at counter waiting for person to return:  20
 Person came back to counter:  21
 Registration process of student Finished:  25
 Waiting for other Student:  25
 Student Arrived at Counter:  28
 Starting Registration process of student:  28
 Going to meet the officer for approvals:  30
 Student at counter waiting for person to return:  30
 Person came back to counter:  31
 Registration process of student Finished:  34
 Waiting for other Student:  34
 Student Arrived at Counter:  37
 Starting Registration process of student:  37
 Going to meet the officer for approvals:  40
 Student at counter waiting for person to return:  40
 Person came back to counter:  41
 Registration process of student Finished:  45
 Waiting for other Student:  45
 Student Arrived at Counter:  48
 Starting Registration process of student:  48
```

Q3. Update the processes created in experiment 2, with number of people performing registration as 2 and number of students as 10.

```
def registration(env,no,wait_duration,regis_duration,per):
```

```
        yield env.timeout(wait_duration)
        print('Student',no,'arrived at Counter:' , env.now)
        with per.request() as req:
            yield req
            print('Starting Registration process of student',no,'at :' , env.now)
            yield env.timeout(regis_duration)
            print("Registration process of student",no,"Finished and"
                  " leaving counter at :",env.now)

    import simpy
    env = simpy.Environment()
    per = simpy.Resource(env, capacity=2)
    n=int(input("Enter number of students: "))
    for i in range(1,n+1):
        env.process(registration(env,i,(i-1)*3,5,per))
    env.run()
```

```
     Student 1 arrived at Counter: 0
     Starting Registration process of student 1 at : 0
     Student 2 arrived at Counter: 3
     Starting Registration process of student 2 at : 3
     Registration process of student 1 Finished and leaving counter at : 5
     Student 3 arrived at Counter: 6
     Starting Registration process of student 3 at : 6
     Registration process of student 2 Finished and leaving counter at : 8
     Student 4 arrived at Counter: 9
     Starting Registration process of student 4 at : 9
     Registration process of student 3 Finished and leaving counter at : 11
     Student 5 arrived at Counter: 12
     Starting Registration process of student 5 at : 12
     Registration process of student 4 Finished and leaving counter at : 14
     Registration process of student 5 Finished and leaving counter at : 17
```

Lab Experiment-4:->

SimPy basics:->

```
import simpy
def example(env):
  event = simpy.events.Timeout(env, delay=1, value=42)
  value = yield event
  print('now=%d, value=%d' % (env.now, value))
env = simpy.Environment()
example_gen = example(env)
p = simpy.events.Process(env, example_gen)
env.run()
```

```
     now=1, value=42
```

Environments :->

Simulation control:->

```
import simpy

def my_proc(env):
    yield env.timeout(1)
    return 'Returning From my_proc Function.'

env = simpy.Environment()
proc = env.process(my_proc(env))
env.run(until=proc)
```

```
 'Returning From my_proc Function.'
```

State access

```
def subfunc(env):
    print(env.active_process)

def my_proc(env):
    while True:
        print(env.active_process)
        subfunc(env)
        yield env.timeout(1)

env = simpy.Environment()
p1 = env.process(my_proc(env))
print(env.active_process)
env.step()
print(env.active_process)
```

```
 None
 <Process(my_proc) object at 0x7f4655c69310>
 <Process(my_proc) object at 0x7f4655c69310>
 None
```

Lab Experiment : 5:->

Event creation:->

Adding Callbacks to an Event:->

```
import simpy
def my_callback(event):
    print('Called back from', event)

env = simpy.Environment()
event = env.event()
event.callbacks.append(my_callback)
event.callbacks
```

```
event.callbacks
```

```
[<function __main__.my_callback>]
```

Usage of Event:->

```
class School:
    def __init__(self, env):
        self.env = env
        self.class_ends = env.event()
        self.pupil_procs = [env.process(self.pupil()) for i in range(3)]
        self.bell_proc = env.process(self.bell())

    def bell(self):
        for i in range(2):
            yield self.env.timeout(45)
            self.class_ends.succeed()
            self.class_ends = self.env.event()
            print()

    def pupil(self):
        for i in range(2):
            print(r' \o/', end='')
            yield self.class_ends

school = School(env)
env.run()
```

```
  \o/ \o/ \o/
  \o/ \o/ \o/
```

Processes are Events too:->

SimPy processes (as created by Process or env.process()) have the nice property of being events, too. That means, that a process can yield another process. It will then be resumed when the other process ends. The event's value will be the return value of that process:

```
def sub(env):
    yield env.timeout(1)
    return 23

def parent(env):
    ret = yield env.process(sub(env))
    return ret

env.run(env.process(parent(env)))
from simpy.util import start_delayed

def sub(env):
```

```
    yield env.timeout(1)
    return 23

def parent(env):
    sub_proc = yield start_delayed(env, sub(env), delay=3)
    ret = yield sub_proc
    return ret

env.run(env.process(parent(env)))
```

```
 23
```

Waiting for Mutiple Events at Once:->

```
from simpy.events import AnyOf, AllOf, Event
events = [Event(env) for i in range(3)]
a = AnyOf(env, events)
b = AllOf(env, events)


def test_condition(env):
    t1, t2 = env.timeout(1, value='spam'), env.timeout(2, value='eggs')
    ret = yield t1 | t2
    assert ret == {t1: 'spam'}

    t1, t2 = env.timeout(1, value='spam'), env.timeout(2, value='eggs')
    ret = yield t1 & t2
    assert ret == {t1: 'spam', t2: 'eggs'}

    e1, e2, e3 = [env.timeout(i) for i in range(3)]
    yield (e1 | e2) & e3
    assert all(e.processed for e in [e1, e2, e3])

proc = env.process(test_condition(env))
env.run()


def fetch_values_of_multiple_events(env):
    t1, t2 = env.timeout(1, value='spam'), env.timeout(2, value='eggs')
    r1, r2 = (yield t1 & t2).values()
    assert r1 == 'spam' and r2 == 'eggs'

proc = env.process(fetch_values_of_multiple_events(env))
env.run()
```

Process Interaction:

Sleeping until woken up

```
from random import seed, randint
```

```
from random import seed, randint
seed(23)

import simpy

class EV:
    def __init__(self, env):
        self.env = env
        self.drive_proc = env.process(self.drive(env))
        self.bat_ctrl_proc = env.process(self.bat_ctrl(env))
        self.bat_ctrl_reactivate = env.event()

    def drive(self, env):
        while True:
            yield env.timeout(randint(20, 40))
            print('Start parking at', env.now)
            self.bat_ctrl_reactivate.succeed() # "reactivate"
            self.bat_ctrl_reactivate = env.event()
            yield env.timeout(randint(60, 360))
            print('Stop parking at', env.now)

    def bat_ctrl(self, env):
        while True:
            print('Bat. ctrl. passivating at', env.now)
            yield self.bat_ctrl_reactivate # "passivate"
            print('Bat. ctrl. reactivated at', env.now)

            # Intelligent charging behavior here ...
            yield env.timeout(randint(30, 90))

env = simpy.Environment()
ev = EV(env)
env.run(until=150)
```

```
 Bat. ctrl. passivating at 0
 Start parking at 29
 Bat. ctrl. reactivated at 29
 Bat. ctrl. passivating at 60
 Stop parking at 131
```

Waiting for another process to terminate:

```
class EV:
    def __init__(self, env):
        self.env = env
        self.drive_proc = env.process(self.drive(env))

    def drive(self, env):
        while True:
            # Drive for 20-40 min
            yield env.timeout(randint(20, 40))
```

```
            # Park for 1-6 hours
            print('Start parking at', env.now)
            charging = env.process(self.bat_ctrl(env))
            parking = env.timeout(randint(60, 360))
            yield charging & parking
            print('Stop parking at', env.now)

    def bat_ctrl(self, env):
        print('Bat. ctrl. started at', env.now)
        # Intelligent charging behavior here ...
        yield env.timeout(randint(30, 90))
        print('Bat. ctrl. done at', env.now)

env = simpy.Environment()
ev = EV(env)
env.run(until=310)
```

```
 Start parking at 29
 Bat. ctrl. started at 29
 Bat. ctrl. done at 83
 Stop parking at 305
```

Interrupting another Process:

```
class EV:
    def __init__(self, env):
        self.env = env
        self.drive_proc = env.process(self.drive(env))

    def drive(self, env):
        while True:
            # Drive for 20-40 min
            yield env.timeout(randint(20, 40))

            # Park for 1 hour
            print('Start parking at', env.now)
            charging = env.process(self.bat_ctrl(env))
            parking = env.timeout(60)
            yield charging | parking
            if not charging.triggered:
                # Interrupt charging if not already done.
                charging.interrupt('Need to go!')
            print('Stop parking at', env.now)

    def bat_ctrl(self, env):
        print('Bat. ctrl. started at', env.now)
        try:
            yield env.timeout(randint(60, 90))
            print('Bat. ctrl. done at', env.now)
        except simpy.Interrupt as i:
```

```
            except simpy.Interrupt as i:
                # Onoes! Got interrupted before the charging was done.
                print('Bat. ctrl. interrupted at', env.now, 'msg:',i.cause)

env = simpy.Environment()
ev = EV(env)
env.run(until=100)
```

```
     Start parking at 31
     Bat. ctrl. started at 31
     Stop parking at 91
     Bat. ctrl. interrupted at 91 msg: Need to go!
```

Shared Resources:

Resources:

Resources can be used by a limited number of processes at a time (e.g., a gas station with a limited number of fuel pumps). Processes request these resources to become a user (or to “own” them) and have to release them once they are done (e.g., vehicles arrive at the gas station, use a fuel-pump, if one is available, and leave when they are done).

SimPy implements three resource types:

Resource PriorityResource, where queueing processes are sorted by priority PreemptiveResource, where processes additionally may preempt other processes with a lower priority

1. Resource:

The Resource is conceptually a semaphore. Its only parameter – apart from the obligatory reference to an Environment – is its capacity. It must be a positive number and defaults to 1: Resource(env, capacity=1).

Here is a basic example for using a resource:

```
import simpy

def resource_user(env, resource):
    request = resource.request() # Generate a request event
    yield request # Wait for access
    yield env.timeout(1) # Do something
    resource.release(request) # Release the resource

env = simpy.Environment()
res = simpy.Resource(env, capacity=1)
user = env.process(resource_user(env, res))
env.run()
```

In above code we need to release the resource under all conditions; for example, if we get interrupted while waiting for or using the resource. In order to help us with that and to avoid too many try: ... finally: ... constructs, request events can be used as context manager, for example:

```
def resource_user(env, resource):
    with resource.request() as req: # Generate a request event
        yield req # Wait for access
        yield env.timeout(1) # Do something
        # Resource released automatically
user = env.process(resource_user(env, res))
env.run()
```

Resources allow us to retrieve lists of the current users or queued users the number of current users and the resource's capacity

```
res = simpy.Resource(env, capacity=1)

def print_stats(res):
    print(f'{res.count} of {res.capacity} slots are allocated.')
    print(f' Users: {res.users}')
    print(f' Queued events: {res.queue}')

def user(res):
    print_stats(res)
    with res.request() as req:
        yield req
        print_stats(res)
    print_stats(res)

procs = [env.process(user(res)), env.process(user(res))]
env.run()
```

```
0 of 1 slots are allocated.
 Users: []
 Queued events: []
1 of 1 slots are allocated.
 Users: [<Request() object at 0x7f4655597610>]
 Queued events: []
1 of 1 slots are allocated.
 Users: [<Request() object at 0x7f4655597610>]
 Queued events: [<Request() object at 0x7f4655597950>]
0 of 1 slots are allocated.
 Users: []
 Queued events: [<Request() object at 0x7f4655597950>]
1 of 1 slots are allocated.
 Users: [<Request() object at 0x7f4655597950>]
 Queued events: []
```

```
 0 of 1 slots are allocated.
  Users: []
  Queued events: []
```

2. Priority Resource:

This subclass of Resource lets requesting processes provide a priority for each request. More important requests will gain access to the resource earlier than less important ones. Priority is expressed by integer numbers; smaller numbers mean a higher priority.

Apart from that, it works like a normal Resource:

```
def resource_user(name, env, resource, wait, prio):
    yield env.timeout(wait)
    with resource.request(priority=prio) as req:
        print(f'{name} requesting at {env.now} with priority={prio}')
        yield req
        print(f'{name} got resource at {env.now}')
        yield env.timeout(3)

env = simpy.Environment()
res = simpy.PriorityResource(env, capacity=1)
p1 = env.process(resource_user(1, env, res, wait=0, prio=0))
p2 = env.process(resource_user(2, env, res, wait=1, prio=0))
p3 = env.process(resource_user(3, env, res, wait=2, prio=-1))
env.run()
```

```
 1 requesting at 0 with priority=0
 1 got resource at 0
 2 requesting at 1 with priority=0
 3 requesting at 2 with priority=-1
 3 got resource at 3
 2 got resource at 6
```

3. PreemptiveResource:

```
def resource_user(name, env, resource, wait, prio):
    yield env.timeout(wait)
    with resource.request(priority=prio) as req:
        print(f'{name} requesting at {env.now} with priority={prio}')
        yield req
        print(f'{name} got resource at {env.now}')
        try:
            yield env.timeout(3)
        except simpy.Interrupt as interrupt:
            by = interrupt.cause.by
            usage = env.now - interrupt.cause.usage_since
            print(f'{name} got preempted by {by} at {env.now}'f' after {usage}')
```

```
env = simpy.Environment()
res = simpy.PreemptiveResource(env, capacity=1)
p1 = env.process(resource_user(1, env, res, wait=0, prio=0))
p2 = env.process(resource_user(2, env, res, wait=1, prio=0))
p3 = env.process(resource_user(3, env, res, wait=2, prio=-1))
env.run()
```

```
1 requesting at 0 with priority=0
1 got resource at 0
2 requesting at 1 with priority=0
3 requesting at 2 with priority=-1
1 got preempted by <Process(resource_user) object at 0x7f4655c69c90> at 2 after 2
3 got resource at 2
2 got resource at 5
```

The following example shows that preemptive low priority requests cannot queue-jump over high priority requests:

```
def user(name, env, res, prio, preempt):
    with res.request(priority=prio, preempt=preempt) as req:
        try:
            print(f'{name} requesting at {env.now}')
            assert isinstance(env.now, int), type(env.now)
            yield req
            assert isinstance(env.now, int), type(env.now)
            print(f'{name} got resource at {env.now}')
            yield env.timeout(3)
        except simpy.Interrupt:
            print(f'{name} got preempted at {env.now}')

env = simpy.Environment()
res = simpy.PreemptiveResource(env, capacity=1)
A = env.process(user('A', env, res, prio=0, preempt=True))
env.run(until=1) # Give A a head start

B = env.process(user('B', env, res, prio=-2, preempt=False))
C = env.process(user('C', env, res, prio=-1, preempt=True))
env.run()
```

```
A requesting at 0
A got resource at 0
B requesting at 1
C requesting at 1
B got resource at 3
C got resource at 6
```

EXAMPLES:->

Bank Renege

```
! pip install simpy
```

```
Collecting simpy
  Downloading https://files.pythonhosted.org/packages/20/f9/874b0bab83406827db93292a5bbe
Installing collected packages: simpy
Successfully installed simpy-4.0.1
```

```
import random
import simpy

RANDOM_SEED = 40
NEW_CUSTOMERS = 5 # Total number of customers
INTERVAL_CUSTOMERS = 10.0 # Generate new customers roughly every x seconds
MIN_PATIENCE = 1 # Min. customer patience
MAX_PATIENCE = 3 # Max. customer patience

def source(env, number, interval, counter):

  for i in range(number):
    c = customer(env, 'Customer%02d' % i, counter, time_in_bank=12.0)
    env.process(c)
    t = random.expovariate(1.0 / interval)
    yield env.timeout(t)

def customer(env, name, counter, time_in_bank):
  arrive = env.now
  print('%7.4f %s: Here I am' % (arrive, name))
  with counter.request() as req:
    patience = random.uniform(MIN_PATIENCE, MAX_PATIENCE)
# Wait for the counter or abort at the end of our tether
    results = yield req | env.timeout(patience)
    wait = env.now - arrive
    if req in results:
# We got to the counter
      print('%7.4f %s: Waited %6.3f' % (env.now, name, wait))
      tib = random.expovariate(1.0 / time_in_bank)
      yield env.timeout(tib)
      print('%7.4f %s: Finished' % (env.now, name))
    else:
# We reneged
      print('%7.4f %s: RENEGED after %6.3f' % (env.now, name, wait))

# Setup and start the simulation
print('Bank renege')
random.seed(RANDOM_SEED)
env = simpy.Environment()
```

```
Bank renege
```

```
# Start processes and run
counter = simpy.Resource(env, capacity=1)
env.process(source(env, NEW_CUSTOMERS, INTERVAL_CUSTOMERS, counter))
env.run()
```

```
      0.0000 Customer00: Here I am
      0.0000 Customer00: Waited  0.000
      0.3884 Customer00: Finished
      6.1361 Customer01: Here I am
      6.1361 Customer01: Waited  0.000
      9.4549 Customer02: Here I am
     10.8288 Customer03: Here I am
     11.1517 Customer02: RENEGED after  1.697
     12.7103 Customer03: RENEGED after  1.882
     19.2352 Customer01: Finished
     32.0106 Customer04: Here I am
     32.0106 Customer04: Waited  0.000
     33.6800 Customer04: Finished
```

Carwash

Covers: • Waiting for other processes • Resources: Resource The Carwash example is a simulation of a carwash with a limited number of machines and a number of cars that arrive at the carwash to get cleaned.

The carwash uses a Resource to model the limited number of washing machines. It also defines a process for washing a car. When a car arrives at the carwash, it requests a machine. Once it got one, it starts the carwash's wash processes and waits for it to finish. It finally releases the machine and leaves.

The cars are generated by a setup process. After creating an intial amount of cars it creates new car processes after a random time interval as long as the simulation continues.

Importing modules which will be required: *bold text**

```
import random
import simpy


OPTIONS=["Gpay","CARD","Paytm"]
RANDOM_SEED = 42
NUM_MACHINES = 2
WASHTIME = 5
T_INTER = 7
SIM_TIME = 20
COST=10
PROFIT=0
```

A carwash has a limited number of machines (NUM_MACHINES) to clean cars in parallel. Cars have to request one of the machines. When they got one, they can start the washing processes and wait for it to finish (which takes washtime minutes).* **In the Wash Function:**

*It takes a car processes and tries to clean it.

```
class Carwash(object):
    def __init__(self, env, num_machines, washtime):
        self.env = env
        self.machine = simpy.Resource(env, num_machines)
        self.washtime = washtime
    def wash(self, car):
        global PROFIT
        yield self.env.timeout(WASHTIME)
        re=random.randint(50, 99)
        print("Carwash removed %d%% of %s's dirt."%(re, car))
        print("Total Cost for",car,":",COST*re)
        PROFIT=PROFIT+(COST*re)
```

In the car process (each car has a name) arrives at the carwash (cw) and requests a cleaning machine. It then starts the washing process, waits for it to finish and leaves to never come back ...

```
def car(env, name, cw):
    print('%s arrives at the carwash at %.2f.' % (name, env.now))
    with cw.machine.request() as request:
        yield request
        print('%s enters the carwash at %.2f.' % (name, env.now))
        yield env.process(cw.wash(name))
        print(name,"leaves the carwash at",env.now,
              "( Paid Bill Using:",random.choice(OPTIONS),")." )
```

setup Function:

We create a carwash, a number of initial cars and keep creating cars approx. every minutes.

```
def setup(env, num_machines, washtime, t_inter):
    carwash = Carwash(env, num_machines, washtime)
    # Create 4 initial cars
    for i in range(4):
        env.process(car(env, 'Car %d' % i, carwash))
    while True:
        yield env.timeout(random.randint(t_inter - 2, t_inter + 2))
        i += 1
    env.process(car(env, 'Car %d' % i, carwash))
```

To make our simulation more accurate I have added a function which displays current Day and Total Profit at end of the day:

```
def day_count(env):
    global PROFIT
    i=1
    while True:
        if i!=1:
            print("\nDay",i-1,"Summary:")
            print("Total Profit Earned: Rs.",PROFIT)
        print("\n ********   DAY:",i,"Starts  ********   \n")
        i=i+1
        PROFIT=0
        yield env.timeout(12)
```

Now we are ready to simulate:

```
print('\n ----- Carwash -----\n')
random.seed(RANDOM_SEED)
env = simpy.Environment()
env.process(day_count(env))
env.process(setup(env, NUM_MACHINES, WASHTIME, T_INTER))
env.run(until=SIM_TIME)
```

```
      ----- Carwash -----


      ********   DAY: 1 Starts  ********

     Car 0 arrives at the carwash at 0.00.
     Car 1 arrives at the carwash at 0.00.
     Car 2 arrives at the carwash at 0.00.
     Car 3 arrives at the carwash at 0.00.
     Car 0 enters the carwash at 0.00.
     Car 1 enters the carwash at 0.00.
     Carwash removed 97% of Car 0's dirt.
     Total Cost for Car 0 : 970
     Carwash removed 67% of Car 1's dirt.
     Total Cost for Car 1 : 670
     Car 0 leaves the carwash at 5 ( Paid Bill Using: Gpay ).
     Car 1 leaves the carwash at 5 ( Paid Bill Using: Gpay ).
     Car 2 enters the carwash at 5.00.
     Car 3 enters the carwash at 5.00.
     Carwash removed 97% of Car 2's dirt.
     Total Cost for Car 2 : 970
     Carwash removed 56% of Car 3's dirt.
     Total Cost for Car 3 : 560
     Car 2 leaves the carwash at 10 ( Paid Bill Using: Paytm ).
     Car 3 leaves the carwash at 10 ( Paid Bill Using: Paytm ).
```

```
Day 1 Summary:
Total Profit Earned: Rs. 3170

 ********   DAY: 2 Starts  ********
```

Project:

Machine Shop

```
! pip install simpy
```

```
Collecting simpy
  Downloading simpy-4.0.1-py2.py3-none-any.whl (29 kB)
Installing collected packages: simpy
Successfully installed simpy-4.0.1
```

```
import random
import simpy
RANDOM_SEED = 42
PT_MEAN = 10.0 # Avg. processing time in minutes
PT_SIGMA = 2.0 # Sigma of processing time
MTTF = 300.0 # Mean time to failure in minutes
BREAK_MEAN = 1 / MTTF # Param. for expovariate distribution
REPAIR_TIME = 30.0 # Time it takes to repair a machine in minutes
JOB_DURATION = 30.0 # Duration of other jobs in minutes
NUM_MACHINES = 10 # Number of machines in the machine shop
WEEKS = 4 # Simulation time in weeks
SIM_TIME = WEEKS * 7 * 24 * 60 # Simulation time in minutes
COUNT=0
PRO0=0
PRO=0
dd=[]
s1=0
s2=0
s3=0
def profit(COUNT):
  global PRO0,var_1
  var_1=random.randint(1,5)
  print("%d",var_1)
  if var_1==1:
    PRO0=PRO0+(COUNT*50)
    dd.append(PRO0)
  elif var_1==2:
    PRO0=PRO0+(COUNT*10)
    dd.append(PRO0)
  elif var_1==3:
    PRO0=PRO0+(COUNT*100)
    dd.append(PRO0)
```

```
      dd.append(PRO0)
    elif var_1==4:
      PRO0=PRO0+(COUNT*120)
      dd.append(PRO0)
    elif var_1==5:
      PRO0=PRO0+(COUNT*150)
      dd.append(PRO0)
  def time_per_part():
    return random.normalvariate(PT_MEAN, PT_SIGMA)
  def time_to_failure():
    return random.expovariate(BREAK_MEAN)
  class Machine(object):
    def __init__(self, env, name, repairman):
      self.env = env
      self.name = name
      self.parts_made = 0
      self.broken = False
      self.process = env.process(self.working(repairman))
      env.process(self.break_machine())
    def working(self, repairman):
      while True:
        done_in = time_per_part()
        while done_in:
          try:
            start = self.env.now
            yield self.env.timeout(done_in)
            done_in = 0 # Set to 0 to exit while loop.
          except simpy.Interrupt:
            self.broken = True
            done_in -= self.env.now - start # How much time left?
            with repairman.request(priority=1) as req:
              yield req
              yield self.env.timeout(REPAIR_TIME)
            self.broken = False
  # Part is done.
        self.parts_made += 1
    def break_machine(self):
      while True:
        yield self.env.timeout(time_to_failure())
        if not self.broken:
          self.process.interrupt()
  def other_jobs(env, repairman):
    while True:
      done_in = JOB_DURATION
      while done_in:
        with repairman.request(priority=2) as req:
          yield req
          try:
            start = env.now
            yield env.timeout(done_in)
            done_in = 0
          except simpy.Interrupt:
```

```
            done_in -= env.now - start
    def Profit_parts(NUM_MACHINES):
      global PRO,s1,s2,s3
      OPTIONS=["Kalsi Machine","Machine INdia","Golden Machine"]
      var_1=random.choice(OPTIONS)
      if var_1=='Kalsi Machine':
        PRO=PRO+(NUM_MACHINE*50)
        dd.append(PRO)
        s1=s1+NUM_MACHINE
      elif var_1=='Machine INdia':
        PRO=PRO+(NUM_MACHINE*100)
        dd.append(PRO)
        s2=s2+NUM_MACHINE
      elif var_1=='Golden Machine':
        PRO=PRO+(NUM_MACHINES*1000)
        dd.append(PRO)
        s3=s3+NUM_MACHINES

    # Setup and start the simulation
    print('Machine shop')
    random.seed(RANDOM_SEED) # This helps reproducing the results
    # Create an environment and start the setup process
    env = simpy.Environment()
    #PRO0=3250
    repairman = simpy.PreemptiveResource(env, capacity=1)
    machines = [Machine(env, 'Machine %d' % i, repairman)
    for i in range(NUM_MACHINES)]
    env.process(other_jobs(env, repairman))
    # Execute!
    env.run(until=SIM_TIME)
    # Analyis/results
    print('Machine shop results after %s weeks' % WEEKS)
    #print('Machine shop profit ',PRO)
    for machine in machines:
      print('%s made %d parts.' % (machine.name, machine.parts_made))
    Profit_parts(NUM_MACHINES)
    print('Profit gained from the parts: ', PRO0)
    print('\n\nNow we are calculating the profit from the repairing')
    print('From Kalsi Machines :%d'%50)
    print(' From Machine India:%d'%100)
    print('From Golden Machines :%d'%1000)
    print('%d Kalsi.'%s1)
    print('%d INdia.'%s2)
    print('%d Golden.'%s3)
    print('Total profit from repairing  :%d'%dd[-1])


         Machine shop
         Machine shop results after 4 weeks
         Machine 0 made 3251 parts.
         Machine 1 made 3273 parts.
         Machine 2 made 3242 parts.
```

```
Machine 3 made 3343 parts.
Machine 4 made 3387 parts.
Machine 5 made 3244 parts.
Machine 6 made 3269 parts.
Machine 7 made 3185 parts.
Machine 8 made 3302 parts.
Machine 9 made 3279 parts.
Profit gained from the parts:  3250


Now we are calculating the profit from the repairing
From Kalsi Machines :50
 From Machine India:100
From Golden Machines :1000
0 Kalsi.
0 INdia.
10 Golden.
Total profit from repairing  :10000
```

✓ 0s completed at 10:52 AM