**Comparativo de Implementación del Modelo de Simulación "Con Fila"**

## 1. Descripción del modelo

El ejercicio tiene como objetivo comparar el comportamiento de un sistema de atención de pasajeros **con fila de espera**, implementado tanto en **Wolfram Mathematica** como en **Python**, utilizando estructuras equivalentes. El sistema simula llegadas, atención y salidas de pasajeros bajo un tiempo máximo (T_{max}) y un servidor único (aforador).

Ambas versiones generan de forma **pseudoaleatoria** los tiempos entre llegadas (TI) y los tiempos de atención (TA), y mantienen un registro de los eventos del sistema (entradas, salidas, tiempos de espera y atención).

## 2. Implementación en Wolfram Mathematica

El siguiente fragmento muestra la estructura del modelo original implementado en Wolfram Mathematica:

```
In[28]:= ClearAll;
         borra todo
         TM = 0; (*Tiempo del modelo en minutos*)
         n = 1500;

         TMax = 60 * 60;

         c = 5;

         TI = Table[1 + Mod[Fibonacci[i], 8], {i, n}]; (*Tiempos entre pasajeros:interarrivo*)
              tabla      op··· sucesión de Fibonacci
         TA = Table[5 + Mod[Prime[i], 10], {i, n}]; (*Tiempos de atención*)
              tabla      op··· número primo
         pasajero = 1;

         ocupado = False; (*Status del aforador*)
                   falso
         llegada = TI[[1]];

         salida = ∞;

         ta = 1; (*Contador para tiempos de atención*)
         eventos = {}; (*Mi lista de eventos*)
         (*Un evento es un {pasajero,llegada,salida}*)
```

In[41]:=
```
While[TM ≤ TMax,
 (*mientras*)
 If[llegada < salida, (*Procesamos una llegada*)
 (*si*)
  TM = llegada;
  (*Print["Llegó el pasajero ",pasajero," en el Min. ",llegada];*)If[¬ ocupado, (*El aforador está desocupado*)
  (*si*)
   ocupado = True;
   (*verdadero*)
   salida = TM + TA[[ta]];
   ta++;
   (*Print["Atendiendo al pasajero ",pasajerosiendoatendido=pasajero];*)evento = {pasajero, llegada, salida};, (*El aforador está ocupado*)(*Print["El pasajero ",pasajero," no es revisado"];
   evento={pasajero,llegada,llegada};*)
   evento = {pasajero, llegada, llegada};];
  (*Programamos la llegada del próximo pasajero*)eventos = Join[eventos, {evento}];
  (*junta*)
  pasajero++;
  llegada = TM + TI[[pasajero]];, (*Procesamos una salida*)
  TM = salida;
  (*Print["Sale el pasajero ",pasajerosiendoatendido," en el minuto ",TM];*)ocupado = False;
  (*falso*)
  salida = ∞;];]
(*Fin del Tiempo de Sim.*)
```

In[42]:=
```
eventos
```

## 3. Implementación en Python

La siguiente traducción en Python conserva la lógica estructural:

```python
import random
from collections import deque

# ----------------------------
# CONFIGURACIÓN INICIAL
# ----------------------------
TMax = 1200                     # Tiempo máximo de simulación (minutos)
TM = 0                          # Tiempo actual del modelo
n = 500                         # Número máximo de pasajeros

# Generación pseudoaleatoria de tiempos
TI = [random.randint(1, 8) for _ in range(n)]        # Tiempos entre llegadas
TA = [random.randint(5, 15) for _ in range(n)]       # Tiempos de atención

# Variables del sistema
pasajero = 1
ocupado = False                 # Estado del aforador
llegada = TI[0]                 # Tiempo de llegada del primer pasajero
salida = float('inf')           # Tiempo de salida inicial (ninguno)
ta = 0                          # Contador de tiempos de atención

# Estructuras de datos
eventos = []                    # Lista de eventos
fila = deque()                  # Cola de espera

# ----------------------------
# SIMULACIÓN CON FILA
# ----------------------------
print("\n--- SIMULACIÓN CON FILA ---")

while TM ≤ TMax:
    # Llega un pasajero
    if llegada < salida:
        TM = llegada

        if not ocupado:
            # Aforador desocupado → atender directamente
            ocupado = True
            salida = TM + TA[ta]
            ta += 1
            evento = ("Entrada y Atención", pasajero, TM, salida, "Atendido")
        else:
            # Aforador ocupado → pasajero entra a la fila
            fila.append(pasajero)
            evento = ("Entrada", pasajero, TM, TM, "No Atendido")

        eventos.append(evento)
        pasajero += 1

        # Programar próxima llegada
        if pasajero < n:
            llegada = TM + TI[pasajero - 1]
        else:
            break
    # Sale un pasajero
    else:
        TM = salida
        evento = ("Salida", "-", TM, salida, "Atendido")
        eventos.append(evento)

        if fila:
            # Hay pasajeros esperando → atender siguiente
            siguiente = fila.popleft()
            salida = TM + TA[ta]
            ta += 1
            evento = ("Atención desde fila", siguiente, TM, salida, "Atendid
o")          eventos.append(evento)
        else:
            # Sin fila → aforador libre
            ocupado = False
            salida = float('inf')
```

**4. Correspondencias entre Mathematica y Python**

| Concepto | Mathematica | Python |
|---|---|---|
| Tiempo máximo de simulación | TMax = 60*60 | TMax = 1200 |
| Generación de TI | Mod[Fibonacci[i], 8] | random.randint(1, 8) |
| Generación de TA | Mod[Prime[i], 10] | random.randint(5, 15) |
| Control de ocupación | If[!ocupado, ... , ...] | if not ocupado: ... else: ... |
| Registro de eventos | eventos = Join[eventos, {evento}] | eventos.append(evento) |
| Fila de espera | No explícita (implícita en lógica) | fila = deque() |
| Ciclo principal | While[TM <= TMax, ...] | while TM <= TMax: |
| Cálculo de tiempos promediO | Mean[salida - llegada] (manual) | sum(duraciones)/len(duraciones) |