Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Clase 19: POO aplicado a Simulación

Rodrigo Toro Icarte (rntoro@uc.cl)

IIC1103 Introducción a la Programación - Sección 5

27 de Mayo, 2015

Simulación

Alguna vez les dije: *"En computación aspiramos a modelar el mundo para solucionar problemas reales"*.

Simulación

Alguna vez les dije: *"En computación aspiramos a modelar el mundo para solucionar problemas reales"*.

Simulación

Objetivo: Aprender a *modelar* problemas reales y *simularlos*.

Simulación

Objetivo: Aprender a *modelar* problemas reales y *simularlos*.

Ejemplos:

- Simular el flujo de personas:
  - En el metro.
  - En el banco.
  - En un tsunami.

Simulación

Objetivo: Aprender a *modelar* problemas reales y *simularlos*.

Ejemplos:

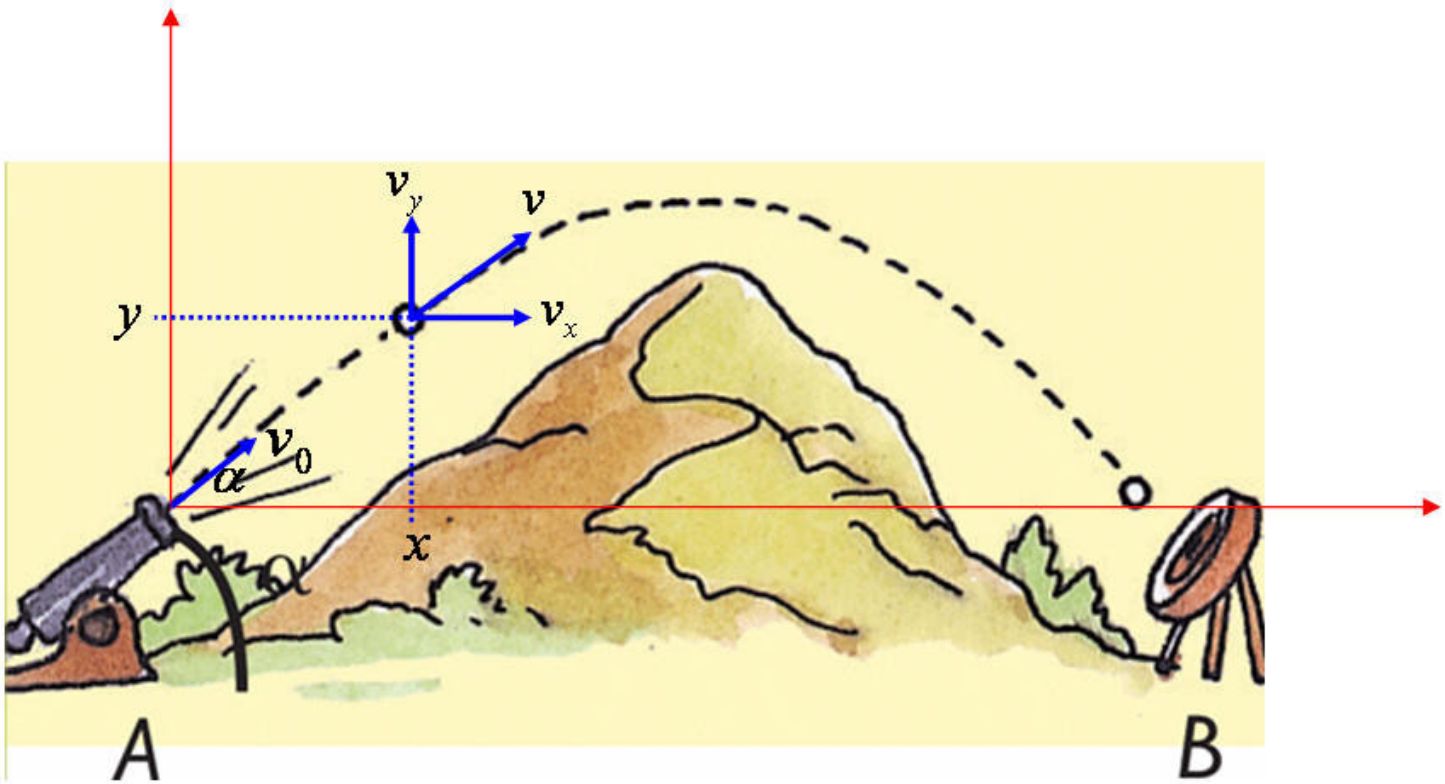
- Simular el flujo de personas:
  - En el metro.
  - En el banco.
  - En un tsunami.
- Simular caudal cuenca hidráulica.
- Simular funcionamiento de una fábrica.

Simulación

Observación: Un *modelo* es una simplificación de la realidad.

Simulación

Observación: Un *modelo* es una simplificación de la realidad.



... mientras más parecido, más confiables serán las conclusiones.

Simulación

Hoy en día se obtienen modelos de simulación muy preciosos (entre **80%** y **95%**).

Simulación

Hoy en día se obtienen modelos de simulación muy preciosos (entre **80%** y **95%**).

Simula UC

http://www.simula.uc.cl/index.php

Simulación

Idea: Tendremos un loop principal que haga avanzar la simulación paso a paso (generalmente es minuto a minuto).

Dentro del loop, ocurrirán o no *eventos* con cierta probabilidad.

Los *eventos* cambiarán nuestras métricas de rendimiento.

Simulación

Ejemplo: ¿Cuál es la espera media en la cola del cajero del metro San Joaquín entre las 19.00 y las 21.00?

Simulación

Ejemplo: ¿Cuál es la espera media en la cola del cajero del metro San Joaquín entre las 19.00 y las 21.00?

Datos históricos:

- En cada minuto, llega una persona al cajero con probabilidad 0.2.
- Cada persona demora entre 1 y 5 minutos usando el cajero (distribuye uniforme).

Simulación

Primero hago un loop para que avance el tiempo (de minuto en minuto).

```
1 def simular():
2   T = 0
3   while(T < 2*60): # simulamos 2 horas
4     # Aquí realizo simulación
5     T += 1
```

Simulación

Primero hago un loop para que avance el tiempo (de minuto en minuto).

```
1 def simular():
2   T = 0
3   while(T < 2*60): # simulamos 2 horas
4     # Aquí realizo simulación
5     T += 1
```

En cada minuto llega alguien con probabilidad 0.2 ¿Cómo modelamos esto?

Simulación

¿Cómo modelo que cierto evento ocurra con probabilidad p?

Simulación

¿Cómo modelo que cierto evento ocurra con probabilidad p?

En código:

```
1 import random
2 def sucede_evento(p):
3   return random.random() < p
4
5 print(sucede_evento(0))    # Siempre False
6 print(sucede_evento(0.7)) # 70% de las veces True
7 print(sucede_evento(1))    # Siempre True
```

Simulación

Agreguemos esta idea el código:

```
1 from random import random
2
3 def simular():
4   T = 0
5   while(T < 2*60): # simulamos 2 horas
6     if(random() < 0.2):
7       # Llega una persona...
8     T += 1
```

Simulación

Agreguemos esta idea el código:

```
1 from random import random
2
3 def simular():
4   T = 0
5   while(T < 2*60): # simulamos 2 horas
6     if(random() < 0.2):
7       # Llega una persona...
8     T += 1
```

¿Qué pasa cuando llega una persona?

Simulación

Agreguemos esta idea el código:

```
1 from random import random
2
3 def simular():
4   T = 0
5   while(T < 2*60): # simulamos 2 horas
6     if(random() < 0.2):
7       # Llega una persona...
8     T += 1
```

¿Qué pasa cuando llega una persona?

- Usa el cajero.
- Se agrega a la cola.

Simulación

Creemos una variable que recuerde la cantidad de personas en cola (que incluya a quien usa el cajero).

Simulación

Creemos una variable que recuerde la cantidad de personas en cola (que incluya a quien usa el cajero).

```
1 from random import random
2
3 def simular():
4   T = 0
5   en_cola = 0
6   while(T < 2*60): # simulamos 2 horas
7     if(random() < 0.2):
8       en_cola += 1
9     T += 1
```

Simulación

Creemos una variable que recuerde la cantidad de personas en cola (que incluya a quien usa el cajero).

```
1 from random import random
2
3 def simular():
4   T = 0
5   en_cola = 0
6   while(T < 2*60): # simulamos 2 horas
7     if(random() < 0.2):
8       en_cola += 1
9     T += 1
```

¿Cómo modelamos que *"el cajero se demora uniforme entre 1 y 5 minutos."*?

Simulación

¿Cómo obtengo valor aleatorio uniforme entre `a` y `b`?

Simulación

¿Cómo obtengo valor aleatorio uniforme entre `a` y `b`?

Caso discreto:

```
1 import random
2 def obtener_numero(a,b):
3   return random.randint(a,b)
```

Caso continuo:

```
1 import random
2 def obtener_numero(a,b):
3   return a + (b-a) * random.random()
```

Simulación

Idea: Cuando alguien comience a usar el cajero, fijemos su tiempo de *ida*.

Simulación

Idea: Cuando alguien comience a usar el cajero, fijemos su tiempo de *ida*.

```
1  from random import random, randint
2  def simular():
3    T = 0
4    T_ida = -1 # <- tiempo en que se desocupa el cajero
5    en_cola = 0
6    while(T < 2*60):
7      if(random() < 0.2):
8        en_cola += 1
9        #reviso si puedo usar el cajero
10       if(en_cola == 1):
11         # Fijo tiempo en que cajero de desocupará
12         T_ida = randint(1,5)
13     T += 1
14     # disminuyo tiempo de ida
15     T_ida -= 1
```

Simulación

Si `T_ida` llega a cero, alguien se va...

Simulación

Si `T_ida` llega a cero, alguien se va...

```
1  from random import random, randint
2  def simular():
3    T = 0; T_ida = -1; en_cola = 0
4    while(T < 2*60):
5      if(random() < 0.2):
6        en_cola += 1
7        if(en_cola == 1):
8          T_ida = randint(1,5)
9      if(T_ida == 0):
10       en_cola -= 1 # alguien se va
11       if(en_cola > 0): # alguien más usa el cajero
12         T_ida = randint(1,5)
13     T += 1
14     T_ida -= 1
```

Simulación

Si `T_ida` llega a cero, alguien se va...

```
from random import random, randint
def simular():
  T = 0; T_ida = -1; en_cola = 0
  while(T < 2*60):
    if(random() < 0.2):
      en_cola += 1
      if(en_cola == 1):
        T_ida = randint(1,5)
    if(T_ida == 0):
      en_cola -= 1 # alguien se va
      if(en_cola > 0): # alguien más usa el cajero
        T_ida = randint(1,5)
    T += 1
    T_ida -= 1
```

¿Qué estadísticas interesantes podemos sacar de este código?

Simulación

¿Cómo calculamos el tiempo medio de espera en cola?

```
1  from random import random, randint
2  def simular():
3    T = 0; T_ida = -1; en_cola = 0
4    while(T < 2*60):
5      if(random() < 0.2):
6        en_cola += 1
7        if(en_cola == 1):
8          T_ida = randint(1,5)
9      if(T_ida == 0):
10       en_cola -= 1 # alguien se va
11       if(en_cola > 0): # alguien más usa el cajero
12         T_ida = randint(1,5)
13     T += 1
14     T_ida -= 1
```

Simulación

¿Cómo calculamos el tiempo medio de espera en cola?

```
1  from random import random, randint
2  def simular():
3    T = 0; T_ida = -1; en_cola = 0
4    while(T < 2*60):
5      if(random() < 0.2):
6        en_cola += 1
7        if(en_cola == 1):
8          T_ida = randint(1,5)
9      if(T_ida == 0):
10       en_cola -= 1 # alguien se va
11       if(en_cola > 0): # alguien más usa el cajero
12         T_ida = randint(1,5)
13     T += 1
14     T_ida -= 1
```

... necesitamos recordar la hora a la que llega cada usuario.

Simulación

Convirtamos la cola en una lista con los tiempos de llegada.

Simulación

Convirtamos la cola en una lista con los tiempos de llegada.

```
1  from random import random, randint
2  def simular():
3    T = 0; T_ida = -1; cola = []
4    while(T < 2*60):
5      if(random() < 0.2):
6        cola.append(T) # <- agrego tiempo de llegada
7        if(len(cola) == 1):
8          T_ida = randint(1,5)
9      if(T_ida == 0):
10       # saco a la persona de la cola
11       cola.remove(cola[0])
12       if(len(cola) > 0):
13         T_ida = randint(1,5)
14     T += 1
15     T_ida -= 1
```

Simulación

Convirtamos la cola en una lista con los tiempos de llegada.

```
1  from random import random, randint
2  def simular():
3    T = 0; T_ida = -1; cola = []
4    while(T < 2*60):
5      if(random() < 0.2):
6        cola.append(T) # <- agrego tiempo de llegada
7        if(len(cola) == 1):
8          T_ida = randint(1,5)
9      if(T_ida == 0):
10       # saco a la persona de la cola
11       cola.remove(cola[0])
12       if(len(cola) > 0):
13         T_ida = randint(1,5)
14     T += 1
15     T_ida -= 1
```

... Así, cuando alguien tome el cajero, calculamos cuánto estuvo en cola.

Simulación

```
def simular():
  T = 0; T_ida = -1; cola = []
  espera = 0     # <- guardo la espera
  atendidos = 0 # <- número de personas atendidas
  while(T < 2*60):
    if(random() < 0.2):
      cola.append(T)
      if(len(cola) == 1):
        T_ida = randint(1,5)
        # agrego tiempo de espera
        atendidos += 1
    if(T_ida == 0):
      cola.remove(cola[0])
      if(len(cola) > 0):
        T_ida = randint(1,5)
        # agrego tiempo de espera
        espera += T - cola[0]
        atendidos += 1
    T += 1; T_ida -= 1
```

Simulación

Finalmente retornamos la espera media en cola:

Simulación

Finalmente retornamos la espera media en cola:

```
1  from random import random, randint
2  def simular():
3    T = 0; T_ida = -1; cola = []
4    espera = 0; atendidos = 0
5    while(T < 2*60):
6      if(random() < 0.2):
7        cola.append(T)
8        if(len(cola) == 1):
9          T_ida = randint(1,5)
10         atendidos += 1
11     if(T_ida == 0):
12       cola.remove(cola[0])
13       if(len(cola) > 0):
14         T_ida = randint(1,5)
15         espera += T - cola[0]
16         atendidos += 1
17     T += 1; T_ida -= 1
18   return espera/atendidos # Retorno
```

Simulación

Ahora podemos ver el resultado de simular:

```
26 print("espera media",simular())
```

Simulación

Ahora podemos ver el resultado de simular:

```
26 print("espera media",simular())
```

¿Es este resultado estable?

Simulación

Para obtener el resultado final se calcula el promedio de ejecutar varias replicas (mientras más mejor).

Simulación

Para obtener el resultado final se calcula el promedio de ejecutar varias replicas (mientras más mejor).

```
20 def obtener_promedio(n):
21   res = 0
22   for i in range(n):
23     res += simular()
24   return res/n
```

Luego llamamos a la función:

```
27 print("espera media",obtener_promedio(10000))
```

Simulación

Ejemplo: *Simule un ataque zombie.*

Simulación

Reglas:

- Un humano y un zombie se encuentran con probabilidad 0.05 cada minuto.
- La probabilidad de que el zombie gane el encuentro es 0.4
- La probabilidad de que el humano mate al zombie es 0.2
- La probabilidad de que el humano escape es 0.4

Simulación

```python
import random
def simular(t_min):
  # Variables iniciales
  zombies = 10; humanos = 10000; T = 0
  while(T < t_min):
    # Un zombie y un humano se encuentran
    if(random.random() < 0.05):
      r = random.random()
      if(r < 0.4):    # Gana zombie
        humanos -= 1
        zombies += 1
      elif(r < 0.6): # Gana humano
        zombies -= 1
    # Salgo si humanos o zombies llegan a cero
    if(zombies == 0 or humanos == 0): break
    # aumento reloj
    T += 1
```

Simulación

Al finalizar muestro estadísticas:

```
18    # Muestro estadísticas
19    print("Días transcurridos:",T//(60*24))
20    print("Humanos:", humanos)
21    print("Zombies:", zombies)
```

Simulo 2 años:

```
23  simular(60*24*365*2)
```

Simulación

[**Ex 2014-2**]:
El ministro Gómez Lobo te ha pedido que implementes una simulación para determinar la necesidad de buses en los recorridos del Transantiago. Cada recorrido tiene una cantidad determinada de paraderos; la separación entre paraderos es siempre la misma para un recorrido; y la velocidad de las micros fluctúa entre los 100 y 900 metros por minuto, y cambia minuto a minuto. Interesa saber la cantidad máxima y el promedio de micros en un recorrido dado, entre 8.00 AM y 8.00 PM.

Simulación

Para simularlo debes implementar dos clases: `Micro` y `Recorrido`.

Simulación

Para simularlo debes implementar dos clases: `Micro` y `Recorrido`.

La clase `Micro` tiene como atributos:

- `velocidad`
- `ultimo_paradero`: Que representa el último paradero visitado.
- `distancia_del_paradero`: Que corresponde a los metros recorridos después de pasar por el `ultimo_paradero`.

Simulación

Para simularlo debes implementar dos clases: `Micro` y `Recorrido`.

La clase `Micro` tiene como atributos:

- `velocidad`
- `ultimo_paradero`: Que representa el último paradero visitado.
- `distancia_del_paradero`: Que corresponde a los metros recorridos después de pasar por el `ultimo_paradero`.

La clase `Recorrido` tiene como atributos:

- `numero_de_paraderos`
- `distancia_entre_paraderos`
- `micros`: Lista con las micros en el recorrido.

Simulación

a) Implementa los constructores de ambas clases. El de la clase `Micro` debe asignar a las micros una velocidad al azar entre 100 y 900 metros (considera solo números enteros) y comenzar justo en el paradero 1. El de `Recorrido` debe comenzar sin micros y recibir el valor de sus otros atributos como parámetro.

Simulación

Primero definimos nuestra clase `Micro`:

```
1 class Micro:
```

Simulación

Primero definimos nuestra clase `Micro`:

```
1 class Micro:
```

Luego defino el constructor:

```
1 class Micro:
2   def __init__(self):
```

Simulación

Primero definimos nuestra clase `Micro`:

```
1 class Micro:
```

Luego defino el constructor:

```
1 class Micro:
2   def __init__(self):
```

¿Qué pongo en el constructor?
¿Qué parámetros debiera recibir?

Simulación

La clase `Micro` tiene como atributos:

- `velocidad`.
- `ultimo_paradero`.
- `distancia_del_paradero`.

Simulación

La clase `Micro` tiene como atributos:

- `velocidad`.
- `ultimo_paradero`.
- `distancia_del_paradero`.

"El constructor de la clase `Micro` *debe asignar a las micros una velocidad al azar entre* 100 *y* 900 *metros (considera solo números enteros) y comenzar justo en el paradero* 1*".*

Simulación

La clase `Micro` tiene como atributos:

- `velocidad`.
- `ultimo_paradero`.
- `distancia_del_paradero`.

"El constructor de la clase `Micro` *debe asignar a las micros una velocidad al azar entre* 100 *y* 900 *metros (considera solo números enteros) y comenzar justo en el paradero* 1*".*

¿Qué pongo en el constructor?
¿Qué parámetros debiera recibir?

Simulación

Debe iniciar en el primer paradero:

```
1 class Micro:
2   def __init__(self):
3     self.ultimo_paradero = 1
4     self.distancia_del_paradero = 0
5     self.velocidad = #???
```

Simulación

Debe iniciar en el primer paradero:

```
1 class Micro:
2   def __init__(self):
3     self.ultimo_paradero = 1
4     self.distancia_del_paradero = 0
5     self.velocidad = #???
```

¿Cómo asigno a velocidad un número random entre 100 y 900?

Simulación

Debe iniciar en el primer paradero:

```
1 class Micro:
2   def __init__(self):
3     self.ultimo_paradero = 1
4     self.distancia_del_paradero = 0
5     self.velocidad = #???
```

¿Cómo asigno a velocidad un número random entre 100 y 900?

... necesitamos `random.randint(100,900)`

Simulación

Asignamos la velocidad:

```
1 import random
2
3 class Micro:
4   def __init__(self):
5     self.ultimo_paradero = 1
6     self.distancia_del_paradero = 0
7     self.velocidad = random.randint(100,900)
```

Simulación

Asignamos la velocidad:

```
1  import random
2
3  class Micro:
4    def __init__(self):
5      self.ultimo_paradero = 1
6      self.distancia_del_paradero = 0
7      self.velocidad = random.randint(100,900)
```

... con esto ya podríamos crear una micro y ver sus atributos:

```
9  m = Micro()
10 print(m.ultimo_paradero)          # >>> 1
11 print(m.distancia_del_paradero)   # >>> 0
12 print(m.velocidad)                # >>> entre 100 y 900
```

Simulación

Ahora creemos la clase `Recorrido`:

```
9  class Recorrido:
10   def __init__(self):
```

Simulación

Ahora creemos la clase `Recorrido`:

```
9  class Recorrido:
10   def __init__(self):
```

¿Qué pongo en el constructor?
¿Qué parámetros debiera recibir?

Simulación

La clase `Recorrido` tiene como atributos:

- `numero_de_paraderos`
- `distancia_entre_paraderos`
- `micros`: Lista con las micros en el recorrido.

Simulación

La clase `Recorrido` tiene como atributos:

- `numero_de_paraderos`
- `distancia_entre_paraderos`
- `micros`: Lista con las micros en el recorrido.

"El constructor de ***`Recorrido`*** *debe comenzar sin micros y recibir el valor de sus otros atributos como parámetro".*

Simulación

La clase `Recorrido` tiene como atributos:

- `numero_de_paraderos`
- `distancia_entre_paraderos`
- `micros`: Lista con las micros en el recorrido.

"El constructor de `Recorrido` *debe comenzar sin micros y recibir el valor de sus otros atributos como parámetro".*

¿Qué pongo en el constructor?
¿Qué parámetros debiera recibir?

Simulación

Las micros comienzan como una lista vacía:

```
9  class Recorrido:
10   def __init__(self):
11     self.micros=[]
```

Simulación

Las micros comienzan como una lista vacía:

```
9  class Recorrido:
10   def __init__(self):
11     self.micros=[]
```

El resto se reciben como parámetro:

```
9  class Recorrido:
10   def __init__(self, np, dp):
11     self.micros=[]
12     self.numero_de_paraderos = np
13     self.distancia_entre_paraderos = dp
```

Simulación

b) Para la clase `Micro` debes implementar un método que recalcula la velocidad de la micro como un entero al azar entre 100 y 900.

Simulación

Agrego el método a Micro:

```
3 class Micro:
4   def __init__(self):
5     self.ultimo_paradero = 1
6     self.distancia_del_paradero = 0
7     self.velocidad = random.randint(100,900)
8   def recalcular_velocidad(self):
```

Simulación

Agrego el método a Micro:

```
3 class Micro:
4   def __init__(self):
5     self.ultimo_paradero = 1
6     self.distancia_del_paradero = 0
7     self.velocidad = random.randint(100,900)
8   def recalcular_velocidad(self):
```

¿Qué parámetros debiera recibir?
¿Qué pongo en el código?

Simulación

"*recalcula la velocidad de la micro como un entero al azar entre* 100 *y* 900"

Simulación

"*recalcula la velocidad de la micro como un entero al azar entre* 100 *y* 900"

... luego no necesito parámetros externos.

```
3 class Micro:
4   def __init__(self):
5     self.ultimo_paradero = 1
6     self.distancia_del_paradero = 0
7     self.velocidad = random.randint(100,900)
8   def recalcular_velocidad(self):
```

Simulación

¿Cómo actualizo el valor de la velocidad?

Simulación

¿Cómo actualizo el valor de la velocidad?

¿Basta con `velocidad = random.randint(100,900)`?

Simulación

¿Cómo actualizo el valor de la velocidad?

¿Basta con `velocidad = random.randint(100,900)`?

Debemos usar el `self` para cambiar el valor del atributo.

```
3 class Micro:
4   def __init__(self):
5     self.ultimo_paradero = 1
6     self.distancia_del_paradero = 0
7     self.velocidad = random.randint(100,900)
8   def recalcular_velocidad(self):
9     self.velocidad = random.randint(100,900)
```

Simulación

c) Para la clase `Recorrido` deberás implementar tres métodos.

i) Un método `agregar_micro` que agregue una micro en el primer paradero del recorrido.

Simulación

Agregamos el método:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self):
```

Simulación

Agregamos el método:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self):
```

¿Qué parámetros debiera recibir?
¿Qué pongo en el código?

Simulación

Debemos recibir una *micro* como parámetro:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
```

Simulación

Debemos recibir una *micro* como parámetro:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
```

¿Cómo agregamos la micro a la lista de micros?

¿Basta con `micros.append(m)`?

Simulación

Para obtener/modificar un atributo debemos usar el `self`.

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
17     self.micros.append(m)
```

Simulación

ii) Un método `avanzar_micro` que recibe una micro m y ajusta la posición de la micro m de acuerdo a lo que avanza en un minuto, considerando la posición actual, la velocidad y considerando además que cuando una micro visita un paradero no continúa avanzando ese minuto. Más específicamente, supón que la micro está ℓ metros más adelante del paradero p y que v es la velocidad actual. Si $\ell + v$ es mayor a la distancia entre paraderos, entonces la nueva posición de la micro es justo en el paradero $p + 1$. De lo contrario, se ubicará a $\ell + v$ metros más adelante del paradero p.

Simulación

Primero, creamos el método:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
17     self.micros.append(m)
18   def avanzar_micro(self):
```

Simulación

Primero, creamos el método:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
17     self.micros.append(m)
18   def avanzar_micro(self):
```

¿Qué parámetros debiera recibir?
¿Qué pongo en el código?

Simulación

"Un método `avanzar_micro` *que recibe una micro m"*

Simulación

"Un método ***avanzar_micro*** *que recibe una micro m"*

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
17     self.micros.append(m)
18   def avanzar_micro(self, m):
```

Simulación

"... ajusta la posición de la micro m de acuerdo a lo que avanza en un minuto, considerando la posición actual, la velocidad y considerando además que cuando una micro visita un paradero no continúa avanzando ese minuto."

Simulación

"... ajusta la posición de la micro m de acuerdo a lo que avanza en un minuto, considerando la posición actual, la velocidad y considerando además que cuando una micro visita un paradero no continúa avanzando ese minuto."

Es decir, si micro está a l metro del paradero p, su velocidad es v y la distancia entre paraderos es d ¿Dónde estará en el siguiente minuto?

Simulación

"... ajusta la posición de la micro m de acuerdo a lo que avanza en un minuto, considerando la posición actual, la velocidad y considerando además que cuando una micro visita un paradero no continúa avanzando ese minuto."

Es decir, si micro está a l metro del paradero p, su velocidad es v y la distancia entre paraderos es d ¿Dónde estará en el siguiente minuto?

- Si $v + l < d$, estará a $v + l$ metros de p.
- Si $v + l \geq d$, estará a 0 metros de $p + 1$.

Simulación

Problema: ¿Cómo obtengo l, v, d y p?

- l: Distancia micro al último paradero.
- p: Último paradero de la micro.
- v: Velocidad micro.
- d: Distancia entre paraderos.

Simulación

Problema: ¿Cómo obtengo l, v, d y p?

- l: Distancia micro al último paradero.
- p: Último paradero de la micro.
- v: Velocidad micro.
- d: Distancia entre paraderos.

Son atributos de la micro `m` y del recorrido `self`:

- l: `m.distanca_del_paradero`.
- p: `m.ultimo_paradero`.
- v: `m.velocidad`.
- d: `self.distancia_entre_paraderos`.

Simulación

Calculo nueva posición y pongo condicionales:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
17     self.micros.append(m)
18   def avanzar_micro(self, m):
19     nueva_pos = m.distancia_del_paradero + m.velocidad
20     if(nueva_pos < self.distancia_entre_paraderos):
21       # ajusto posición de la micro
22     else:
23       # Llego al próximo paradero
```

Simulación

Calculo nueva posición y pongo condicionales:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
17     self.micros.append(m)
18   def avanzar_micro(self, m):
19     nueva_pos = m.distancia_del_paradero + m.velocidad
20     if(nueva_pos < self.distancia_entre_paraderos):
21       # ajusto posición de la micro
22     else:
23       # Llego al próximo paradero
```

¿Cómo modifico la posición y paradero de la micro?

Simulación

Modifico posición de la micro:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
17     self.micros.append(m)
18   def avanzar_micro(self, m):
19     nueva_pos = m.distancia_del_paradero + m.velocidad
20     if(nueva_pos < self.distancia_entre_paraderos):
21       m.distancia_del_paradero = nueva_pos
22     else:
23       m.distancia_del_paradero = 0
24       m.ultimo_paradero += 1
```

Simulación

iii) Un método `simular_minuto`, que se encargue de simular un minuto en la simulación: debe avanzar todas las micros, eliminar las que llegan al último paradero y, finalmente, recalcular la velocidad de cada micro para el próximo minuto. Recuerda que puedes eliminar un elemento `e` de una lista `lista` usando `lista.remove(e)`.

Simulación

Creamos el método:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     # (...)
14   def agregar_micro(self, m):
15     # (...)
16   def avanzar_micro(self, m):
17     # (...)
18   def simular_minuto(self):
```

Simulación

Creamos el método:

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     # (...)
14   def agregar_micro(self, m):
15     # (...)
16   def avanzar_micro(self, m):
17     # (...)
18   def simular_minuto(self):
```

¿Qué parámetros debiera recibir?
¿Qué pongo en el código?

Simulación

"...debe avanzar todas las micros, eliminar las que llegan al último paradero y, finalmente, recalcular la velocidad de cada micro para el próximo minuto"

Simulación

"...debe avanzar todas las micros, eliminar las que llegan al último paradero y, finalmente, recalcular la velocidad de cada micro para el próximo minuto"

¿Parámetros?

Simulación

"...debe avanzar todas las micros, eliminar las que llegan al último paradero y, finalmente, recalcular la velocidad de cada micro para el próximo minuto"

¿Parámetros? Ninguno

Simulación

"...debe avanzar todas las micros, eliminar las que llegan al último paradero y, finalmente, recalcular la velocidad de cada micro para el próximo minuto"

¿Parámetros? Ninguno

¿Código?

Simulación

"...debe avanzar todas las micros, eliminar las que llegan al último paradero y, finalmente, recalcular la velocidad de cada micro para el próximo minuto"

¿Parámetros? Ninguno

¿Código?

- Avanzar `todas` las micros.
- Eliminar micros en último paradero.
- Recalcular velocidades.

Simulación

¿Cómo hago avanzar a todas las micros?

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
17     # (...)
18   def avanzar_micro(self, m):
19     # (...)
20   def simular_minuto(self):
```

Simulación

Podemos hacer un `for` sobre `self.micros` y llamar a `avanzar_micro(...)`.

Simulación

Podemos hacer un `for` sobre `self.micros` y llamar a `avanzar_micro(...)`.

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     # (...)
14   def agregar_micro(self, m):
15     # (...)
16   def avanzar_micro(self, m):
17     # (...)
18   def simular_minuto(self):
19     for m in self.micros:
20       # Llamar a avanzar_micro
```

Simulación

Podemos hacer un `for` sobre `self.micros` y llamar a `avanzar_micro(...)`.

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     # (...)
14   def agregar_micro(self, m):
15     # (...)
16   def avanzar_micro(self, m):
17     # (...)
18   def simular_minuto(self):
19     for m in self.micros:
20       # Llamar a avanzar_micro
```

¿Cómo llamo a `avanzar_micro` desde `Recorrido`?
¿Basta con `avanzar_micro(m)`?

Simulación

Para llamar a un método desde dentro de una clase usamos `self`.

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     # (...)
14   def agregar_micro(self, m):
15     # (...)
16   def avanzar_micro(self, m):
17     # (...)
18   def simular_minuto(self):
19     for m in self.micros:
20       self.avanzar_micro(m)
```

Simulación

¿Cómo elimino las micros del último paradero?

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
17     # (...)
18   def avanzar_micro(self, m):
19     # (...)
20   def simular_minuto(self):
21     for m in self.micros:
22       self.avanzar_micro(m)
```

Simulación

¿Cómo elimino las micros del último paradero?

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     self.micros=[]
14     self.numero_de_paraderos = np
15     self.distancia_entre_paraderos = dp
16   def agregar_micro(self, m):
17     # (...)
18   def avanzar_micro(self, m):
19     # (...)
20   def simular_minuto(self):
21     for m in self.micros:
22       self.avanzar_micro(m)
```

... ¿Cómo sé si una micro está en el último paradero?

Simulación

Una micro está en el último paradero si m.ultimo_paradero es igual a `self.numero_de_paraderos`.

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     # (...)
14   def agregar_micro(self, m):
15     # (...)
16   def avanzar_micro(self, m):
17     # (...)
18   def simular_minuto(self):
19     for m in self.micros:
20       self.avanzar_micro(m)
21     for m in self.micros:
22       if(m.ultimo_paradero==self.numero_de_paraderos):
23         # elimino m de la lista de micros
```

Simulación

Una micro está en el último paradero si m.ultimo_paradero es igual a `self.numero_de_paraderos`.

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     # (...)
14   def agregar_micro(self, m):
15     # (...)
16   def avanzar_micro(self, m):
17     # (...)
18   def simular_minuto(self):
19     for m in self.micros:
20       self.avanzar_micro(m)
21     for m in self.micros:
22       if(m.ultimo_paradero==self.numero_de_paraderos):
23         # elimino m de la lista de micros
```

¿Cómo elimino `m`?

Simulación

Basta con `self.micros.remove(m)`

```
11 class Recorrido:
12   def __init__(self, np, dp):
13     # (...)
14   def agregar_micro(self, m):
15     # (...)
16   def avanzar_micro(self, m):
17     # (...)
18   def simular_minuto(self):
19     for m in self.micros:
20       self.avanzar_micro(m)
21     for m in self.micros:
22       if(m.ultimo_paradero==self.numero_de_paraderos):
23         self.micros.remove(m)
```

Simulación

¿Cómo recalculo las velocidades?

```
3  class Micro:
4    # (...)
5    def recalcular_velocidad(self):
6      self.velocidad = random.randint(100,900)
7
8  class Recorrido:
9    # (...)
10   def simular_minuto(self):
11     for m in self.micros:
12       self.avanzar_micro(m)
13     for m in self.micros:
14       if(m.ultimo_paradero==self.numero_de_paraderos):
15         self.micros.remove(m)
```

Simulación

Recorremos las micros y llamamos a recalcular su velocidad.

```
3  class Micro:
4    # (...)
5    def recalcular_velocidad(self):
6      self.velocidad = random.randint(100,900)
7
8  class Recorrido:
9    # (...)
10   def simular_minuto(self):
11     for m in self.micros:
12       self.avanzar_micro(m)
13     for m in self.micros:
14       if(m.ultimo_paradero==self.numero_de_paraderos):
15         self.micros.remove(m)
16     for m in self.micros:
17       # Llamo a recalcular_velocidad
```

Simulación

Recorremos las micros y llamamos a recalcular su velocidad.

```
3  class Micro:
4    # (...)
5    def recalcular_velocidad(self):
6      self.velocidad = random.randint(100,900)
7
8  class Recorrido:
9    # (...)
10   def simular_minuto(self):
11     for m in self.micros:
12       self.avanzar_micro(m)
13     for m in self.micros:
14       if(m.ultimo_paradero==self.numero_de_paraderos):
15         self.micros.remove(m)
16     for m in self.micros:
17       # Llamo a recalcular_velocidad
```

... ¿Cómo llamo a `recalcular_velocidad`?

Simulación

Lo llamo a través de la micro particular.

```
3  class Micro:
4    # (...)
5    def recalcular_velocidad(self):
6      self.velocidad = random.randint(100,900)
7
8  class Recorrido:
9    # (...)
10   def simular_minuto(self):
11     for m in self.micros:
12       self.avanzar_micro(m)
13     for m in self.micros:
14       if(m.ultimo_paradero==self.numero_de_paraderos):
15         self.micros.remove(m)
16     for m in self.micros:
17       m.recalcular_velocidad()
```

Simulación

Lo llamo a través de la micro particular.

```
3  class Micro:
4    # (...)
5    def recalcular_velocidad(self):
6      self.velocidad = random.randint(100,900)
7
8  class Recorrido:
9    # (...)
10   def simular_minuto(self):
11     for m in self.micros:
12       self.avanzar_micro(m)
13     for m in self.micros:
14       if(m.ultimo_paradero==self.numero_de_paraderos):
15         self.micros.remove(m)
16     for m in self.micros:
17       m.recalcular_velocidad()
```

¿Es esto suficiente?

Simulación

Lo llamo a través de la micro particular.

```
3  class Micro:
4    # (...)
5    def recalcular_velocidad(self):
6      self.velocidad = random.randint(100,900)
7
8  class Recorrido:
9    # (...)
10   def simular_minuto(self):
11     for m in self.micros:
12       self.avanzar_micro(m)
13     for m in self.micros:
14       if(m.ultimo_paradero==self.numero_de_paraderos):
15         self.micros.remove(m)
16     for m in self.micros:
17       m.recalcular_velocidad()
```

¿Es esto suficiente? ... Sí.

Simulación

d) Implementa una función `simular` que reciba como parámetros un número n de paraderos, la distancia d entre paraderos y la cantidad $mins$ de minutos que transcurren entre la salida de una micro y la próxima. Esta función debe imprimir el máximo y el promedio de la cantidad de micros que hubo en el recorrido. Cada iteración corresponde a un minuto, y deberás simular por 12 horas. Inicialmente el recorrido comienza con una micro en el primer paradero y se genera una nueva micro en ese paradero cada $mins$ minutos.

Simulación

¿Dónde implemento la función `simular`?

```
1 import random
2 class Micro:
3   # (...)
4 class Recorrido:
5   # (...)
```

Simulación

¿Dónde implemento la función `simular`?

```
1 import random
2 class Micro:
3   # (...)
4 class Recorrido:
5   # (...)
```

... en el código principal:

```
1 import random
2 class Micro:
3   # (...)
4 class Recorrido:
5   # (...)
6
7 def simular():
```

Simulación

¿Qué parámetros recibe?

Simulación

¿Qué parámetros recibe?

"... que reciba como parámetros un número n de paraderos, la distancia d entre paraderos y la cantidad mins de minutos que transcurren entre la salida de una micro y la próxima."

Simulación

¿Qué parámetros recibe?

"... que reciba como parámetros un número n de paraderos, la distancia d entre paraderos y la cantidad mins de minutos que transcurren entre la salida de una micro y la próxima."

```
1 import random
2 class Micro:
3   # (...)
4 class Recorrido:
5   # (...)
6
7 def simular(n,d,mins):
```

Simulación

¿Código de la función?

Simulación

¿Código de la función?

"Esta función debe imprimir el máximo y el promedio de la cantidad de micros que hubo en el recorrido. Cada iteración corresponde a un minuto, y deberás simular por 12 horas. Inicialmente el recorrido comienza con una micro en el primer paradero y se genera una nueva micro en ese paradero cada mins minutos."

Simulación

Código:

- Crear un `Recorrido` y agregarle una `Micro` en el primer paradero.
- `While` que itere por $12 \cdot 60$ minutos.
- Simular recorrido minuto a minuto.
- Agregar una micro cada `mins` minutos.
- Agregar variables que recuerden máximo y promedio micros en recorrido.

Simulación

¿Cómo creo un `Recorrido` con una micro en el paradero 1?

```
1  import random
2  class Micro:
3    def __init__(self):
4      self.ultimo_paradero = 1
5      self.distancia_del_paradero = 0
6      self.velocidad = random.randint(100,900)
7    # (...)
8  class Recorrido:
9    def __init__(self, np, dp):
10     self.micros=[]
11     self.numero_de_paraderos = np
12     self.distancia_entre_paraderos = dp
13   def agregar_micro(self, m):
14     self.micros.append(m)
15   # (...)
16
17 def simular(n,d,mins):
```

Simulación

Crear el recorrido: `r = Recorrido(n,d)`
Crear una micro: `m = Micro()`

Simulación

Crear el recorrido: `r = Recorrido(n,d)`
Crear una micro: `m = Micro()`

¿En qué paradero se crea la micro?

Simulación

Crear el recorrido: `r = Recorrido(n,d)`
Crear una micro: `m = Micro()`

¿En qué paradero se crea la micro?

... como la `m` ya está en el paradero, basta con agregarla al recorrido. ¿Cómo la agrego?

Simulación

Crear el recorrido: `r = Recorrido(n,d)`
Crear una micro: `m = Micro()`

¿En qué paradero se crea la micro?

... como la `m` ya está en el paradero, basta con agregarla al recorrido. ¿Cómo la agrego?

Agregar micro: `r.agregar_micro(m)`

Simulación

Resultado:

```
1  import random
2  class Micro:
3    # (...)
4  class Recorrido:
5    # (...)
6
7  def simular(n,d,mins):
8    r = Recorrido(n,d)
9    m = Micro()
10   r.agregar_micro(m)
```

Simulación

¿Cómo hago un loop por $12 \cdot 60$ minutos?

Simulación

¿Cómo hago un loop por $12 \cdot 60$ minutos?

```
1  import random
2  class Micro:
3    # (...)
4  class Recorrido:
5    # (...)
6
7  def simular(n,d,mins):
8    r = Recorrido(n,d)
9    m = Micro()
10   r.agregar_micro(m)
11   T = 1
12   while(T <= 12*60):
13     # Código de la simulación
14     T += 1
```

Simulación

¿Cómo simulo 1 minuto del recorrido?

```
1  import random
2  class Micro:
3    # (...)
4  class Recorrido:
5    # (...)
6    def simular_minuto(self):
7      # (...)
8  def simular(n,d,mins):
9    r = Recorrido(n,d)
10   m = Micro()
11   r.agregar_micro(m)
12   T = 1
13   while(T <= 12*60):
14     # Código de la simulación
15     T += 1
```

Simulación

Hay que llamar a `simular_minuto` de `Recorrido`
¿Cómo lo llamo?

Simulación

Hay que llamar a `simular_minuto` de `Recorrido`
¿Cómo lo llamo? ... debe ser a través de `r`.

Simulación

Hay que llamar a `simular_minuto` de `Recorrido`
¿Cómo lo llamo? ... debe ser a través de `r`.

```
1  import random
2  class Micro:
3    # (...)
4  class Recorrido:
5    # (...)
6    def simular_minuto(self):
7      # (...)
8  def simular(n,d,mins):
9    r = Recorrido(n,d)
10   m = Micro()
11   r.agregar_micro(m)
12   T = 1
13   while(T <= 12*60):
14     r.simular_minuto()
15     T += 1
```

Simulación

¿Cómo agrego una `Micro` cada `mins` minutos.?

```
#(...)
def simular(n,d,mins):
  r = Recorrido(n,d)
  m = Micro()
  r.agregar_micro(m)
  T = 1
  while(T <= 12*60):
    # aquí debo agregar nueva micro
    r.simular_minuto()
    T += 1
```

Simulación

La condición puede ser si `T%mins == 0`

Simulación

La condición puede ser si `T%mins == 0`

```
#(...)
def simular(n,d,mins):
  r = Recorrido(n,d)
  m = Micro()
  r.agregar_micro(m)
  T = 1
  while(T <= 12*60):
    if(T%mins == 0):
      # aquí agrego micro
    r.simular_minuto()
    T += 1
```

Simulación

La condición puede ser si `T%mins == 0`

```
#(...)
def simular(n,d,mins):
  r = Recorrido(n,d)
  m = Micro()
  r.agregar_micro(m)
  T = 1
  while(T <= 12*60):
    if(T%mins == 0):
      # aquí agrego micro
    r.simular_minuto()
    T += 1
```

¿Cómo agrego una nueva micro al recorrido?

Simulación

Basta con `r.agregar_micro(Micro())`.

Simulación

Basta con `r.agregar_micro(Micro())`.

```
#(...)
def simular(n,d,mins):
  r = Recorrido(n,d)
  m = Micro()
  r.agregar_micro(m)
  T = 1
  while(T <= 12*60):
    if(T%mins == 0):
      r.agregar_micro(Micro())
    r.simular_minuto()
    T += 1
```

Simulación

Tarea: Agregue dos variables que le permitan imprimir el número máximo y promedio de micros en el recorrido.

Ejercicios

1) La probabilidad de que Chile sea campeón del mundo es de un 1% (p = 0.01). Simule cuántos campeonatos del mundo debemos jugar para ganar uno.

2) Realice varias réplicas para el problema 1 y calcule el promedio de mundiales que debemos jugar para ganarlo. La teoría dice que si la probabilidad es 0.01, entonces deberíamos jugar 100 mundiales para ganar uno. Vea si la teoría se cumple en su programa.

3) Para el ejercicio de las micros, calcule el tiempo de espera medio en cada paradero.

Ejercicios

[Ex-2014/2 recuperativo]

La empresa Metro se ha visto en la necesidad de estudiar los tiempos de espera promedio de sus pasajeros para así ver cuál es la mejor combinación de tamaño y rapidez de sus metros. Para ello te han pedido que hagas una simulación que logre determinar el tiempo de espera promedio de cada uno de los pasajeros que llegan a una estación. Un estudio muestra que en cada minuto llegan entre 5 y 10 personas aleatoriamente. Además el estudio muestra que los pasajeros saben donde están las entradas a los carros, por lo que al llegar a la estación se ponen en una de las filas para entrar a algún carro. La elección de la fila la hacen según cual sea más corta. Los carros no vienen vacíos, sino que llegan con un cierto número de pasajeros. Cuando un metro llega a la estación las personas en las filas comienzan a subir al carro al cual pertenece su fila hasta que el carro se llene o hasta que no queden más personas en la fila. Si el carro se llena esperan el próximo metro (no suben a otro carro). También se sabe que si un pasajero encuentra que todas las filas de espera son de tamaño igual o superior a 20 personas, se retira de la estación y se va en micro.

Ejercicios

a) Implementa dos clases: la clase `Metro` y la clase `Estación` junto a su constructor y atributos.

- El constructor de la clase `Metro` recibe dos parámetros, el número de carros del metro y la capacidad máxima de cada uno. Calcula de forma aleatoria el número de personas que viene en cada carro.
- El constructor de la clase `Estación` recibe sólo un parámetro que indica la cantidad de puertas que tienen los metros (se asume que esta cantidad es la misma para todos los metros). Esta clase es la encargada de guardar como atributo la fila de pasajeros, por cada puerta, esperando por subir al metro. Por ejemplo: si un metro tiene 10 carros, habrán 10 filas de a lo más 20 personas esperado por subir (una fila por cada carro).

Ejercicios

b) Implementa los métodos indicados de las clases Metro y Estación. Puedes agregar más métodos en caso que lo desees.

- Para la clase Metro deberás implementar tres métodos. Un método que retorna el número de pasajeros en un determinado carro. Otro método que represente el ingreso de una persona a un carro determinado del metro. Tenga en cuenta que no es necesario conocer la persona de cada carro sino simplemente el número de personas que hay en cada carro. Y el último método que permite bajar del metro a una cantidad aleatoria de personas al llegar el metro a la estación. Las personas bajan primero del carro y luego suben las que quepan.
- Para la clase Estacion deberás implementar un método que permita agregar la hora de llegada de un pasajero a alguna fila.

Ejercicios

c) Teniendo hecho lo anterior escribe la función `simular` que permita simular el comportamiento de esa estación. Esta función recibe el tiempo total de simulación así como también el tiempo en que llega cada Metro y debe retornar el tiempo promedio de espera de todos los pasajeros.