```python
# Batch Scheduling Algorithms by Chip Henderson
# CS7343 - 15 October 2022

from array import array
import numbers
import random
from statistics import mean
import numpy as np


def arrTimeGen(low: int, k: int, size: int) -> array:
    # Generate arrival times based on uniform distribution
    arrivalTimeArr = np.random.randint(low, k, size)
    return arrivalTimeArr

def cpuTimeGen(d: float, v: float, size: int) -> array:
    # Generate CPU total times based on Gaussian distribution
    cpuTimeArr = np.random.normal(d, v, size)
    return cpuTimeArr

def activeProcess(thisArray: array, time: int) -> array:
    # Loop through all processes, set active flags
    for process in thisArray:
        if time >= process[0] and process[2] != 1:
            process[2] = 1
    return thisArray

def srt_activeProcess(thisArray: array, time: int) -> array:
    # Loop through all processes, set active flags specific for SRT
    for process in thisArray:
        if time >= process[0] and process[2] != 1:
            process[2] = 1
            process[1] = np.random.normal(d, v, size=None)
    #Create an index array to sort by arrival time, then CPU time
    indexArray = np.lexsort((thisArray[:,1],thisArray[:,0]))
    thisArray = thisArray[indexArray]
    return thisArray

def fifoQueue(k: int, d: int, v: float, n: int) -> list:
    ttList = []
    t = 0

    # Create an empty array to hold the sim results
    simArray = np.empty((1,n))

    # Create a zero array to be the flag value
    zeroArray = np.zeros((1,n),float)

    # Generates random total CPU times
    cpuTimeArr = cpuTimeGen(d,v,n)

    # Sort the array by arrival time for FIFO
    arrTimeArr = np.sort(arrTimeGen(0,k,n))

    # Merges the two arrays and transposes the result,
    # assigning a CPU time to an arrival time
    simArray = np.vstack((arrTimeArr, cpuTimeArr, zeroArray)).T

    # Loop through each process in the table
```

```
    for process in simArray:
        while t < process[0]: # Increments t for inactive processes
            t += 1
            simArray = activeProcess(simArray, t)

        i = process[1] # Assign total CPU time to iterator
        while i > 0:
            t += 1
            simArray = activeProcess(simArray, t)
            i -= 1

        process[2] = 0 # Set flag to inactive

        ttList.append(t - process[0])
        avg_tt = round(mean(ttList),2)

        # Delete the process from the array once it has completed
        simArray = np.delete(simArray,0,0)

    return avg_tt

def sjfQueue(k: int, d: int, v: float, n: int) -> list:
    ttList = []
    t = 0

    # Create an empty array to hold the sim results
    simArray = np.empty((1,n))

    # Create a zero array to be the flag value
    zeroArray = np.zeros((1,n),float)

    # Get arrays for arrival time and cpu time, both sorted for SJF
    arrTimeArr = np.sort(arrTimeGen(0,k,n))
    cpuTimeArr = np.sort(cpuTimeGen(d,v,n))

    # Merges the two arrays and transposes the result,
    # assigning a CPU time to an arrival time
    simArray = np.vstack((arrTimeArr, cpuTimeArr, zeroArray)).T

    for process in simArray:
        while t < process[0]: # Increments t for inactive processes
            t += 1
            simArray = activeProcess(simArray, t)

        i = process[1] # Assign total CPU time to iterator
        while i > 0:
            t += 1
            simArray = activeProcess(simArray, t)
            i -= 1

        process[2] = 0 # Set flag to inactive

        ttList.append(t - process[0])
        avg_tt = round(mean(ttList),2)

        # Delete the process from the array once it has completed
        simArray = np.delete(simArray,0,0)

    return avg_tt

```

```python
def srtQueue(k: int, d: int, v: float, n: int) -> list:
    ttList = []
    t = 0

    # Create an empty array to hold the sim results
    simArray = np.empty((1,n))

    # Create a zero array to be CPU time and flag value
    zeroArray = np.zeros((2,n),float)

    # Sort the array by arrival tie
    arrTimeArr = np.sort(arrTimeGen(0,k,n))

    # Merges the two arrays and transposes the result,
    # assigning a CPU time to an arrival time
    simArray = np.vstack((arrTimeArr, zeroArray)).T

    for process in simArray:
        while t < process[0]: # Increments t for inactive processes
            t += 1
            simArray = srt_activeProcess(simArray, t)

        process = simArray[0] # Update process values
        i = process[1] # Assign total CPU time to iterator
        while i > 0:
            t += 1
            simArray = srt_activeProcess(simArray, t)
            i -= 1

        process[2] = 0 # Set flag to inactive

        ttList.append(t - process[0])
        avg_tt = round(mean(ttList),2)

        # Delete the process from the array once it has completed
        simArray = np.delete(simArray,0,0)

    return avg_tt

j = 0
k = 1000
d = 1
v = 0.2 * d
n = 100
avg_ttList = []
while j < 100:
    avg_ttList.append(fifoQueue(k,d,v,n))
    j += 1
print(avg_ttList)

j = 0
avg_ttList = []
while j < 100:
    avg_ttList.append(sjfQueue(k,d,v,n))
    j += 1
print(avg_ttList)

j = 0
avg_ttList = []
while j < 100:
```

```
180     avg_ttList.append(srtQueue(k,d,v,n))
181     j += 1
182 print(avg_ttList)
```