**EX. NO:1(A) BASIC I/O PROGRAM - PYTHON USER INPUT**

# using input() to take user input

num = input('Enter a number: ')

print('You Entered:', num)

print('Data type of num:', type(num))

# EX. NO:1(A) BASIC I/O PROGRAM - PYTHON USER INPUT

Enter a number: 10

You Entered: 10

Data type of num: <class 'str'>

**EX. NO:1(B) BASIC I/O PROGRAM – ODD OR EVEN NUMBERS**

print ("Even or odd numbers") num=int(input("Enter any number: ")) if (num% 2==0):

print ("The given number is Even " ) else:

print ("The given number is Odd ")

**OUTPUT:**

**EX. NO:1(B)** **BASIC I/O PROGRAM – ODD OR EVEN NUMBERS**

Even or odd numbers Enter any number: 5 The given number is Odd

Even or odd numbers Enter any number: 4

The given number is Even

**EX. NO:2 SHORTEST JOB FIRST ALGORITHM**

# function for swapping

def swap(b, c):

b, c = c, b

return b, c

#main function starts

print("SHORTEST JOB FIRST SCHEDULLING")

a = []

Cmp\_time = 0

Total\_WT = 0

Total\_TAT = 0

Avg\_WT = 0

Avg\_TAT = 0

n = int(input("Enter the number of processes: "))

print("Enter the Arrival time and Burst time of the process")

print("AT BT")

for i in range(n):

AT, BT = map(int, input().split())

a.append({'id': i+1, 'AT': AT, 'BT': BT})

# if process are arrived at different time

# then sort the process on the basis of AT

for i in range(n):

for j in range(n-i-1):

if a[j]['AT'] > a[j+1]['AT']:

a[j]['id'], a[j+1]['id'] = swap(a[j]['id'], a[j+1]['id'])

a[j]['AT'], a[j+1]['AT'] = swap(a[j]['AT'], a[j+1]['AT'])

a[j]['BT'], a[j+1]['BT'] = swap(a[j]['BT'], a[j+1]['BT'])

if (a[0]['AT']==1):

Cmp\_time = 1

a[0]['WT'] = a[0]['AT']

Cmp\_time = Cmp\_time+a[0]['BT']

a[0]['TAT'] = Cmp\_time - a[0]['AT']

Total\_WT = Total\_WT + a[0]['WT']

Total\_TAT = Total\_TAT + a[0]['TAT']

for i in range(1, n):

mini = a[i]['BT']

for j in range(i, n):

if mini > a[j]['BT'] and a[j]['AT'] <= Cmp\_time:

mini = a[j]['BT']

a[i]['id'], a[j]['id'] = swap(a[i]['id'], a[j]['id'])

a[i]['AT'], a[j]['AT'] = swap(a[i]['AT'], a[j]['AT'])

a[i]['BT'], a[j]['BT'] = swap(a[i]['BT'], a[j]['BT'])

# completion time of the process

Cmp\_time = Cmp\_time + a[i]['BT']

# Turn Around Time of the process

# compl - Arrival

a[i]['TAT'] = Cmp\_time - a[i]['AT']

# Waiting Time of the process

# TAT - BT

a[i]['WT'] = a[i]['TAT'] - a[i]['BT']

Total\_WT = Total\_WT + a[i]['WT']

Total\_TAT = Total\_TAT + a[i]['TAT']

Avg\_WT = Total\_WT / n

Avg\_TAT = Total\_TAT / n

# Printing of the results

print("ID\tCAT\tTAT\tWT")

for i in range(n):

Cmp\_time = a[i]['TAT'] + a[i]['AT']

print(f"{a[i]['id']}\t{Cmp\_time}\t{a[i]['TAT']}\t{a[i]['WT']}")

print(f"Avg turnaround time is: {Avg\_TAT:.2f}")

print(f"Avg waiting time is: {Avg\_WT:.2f}")

**OUTPUT**

# EX. NO:2 SHORTEST JOB FIRST ALGORITHM

SHORTEST JOB FIRST SCHEDULLING

Enter the number of processes: 6

Enter the Arrival time and Burst time of the process

AT BT

0 7

1 5

2 3

3 1

4 2

5 1

ID CAT TAT WT

1 7 7 0

4 8 5 4

6 9 4 3

5 11 7 5

3 14 12 9

2 19 18 13

Avg turnaround time is: 8.83

Avg waiting time is: 5.67

**EX. NO:3 FIRST COME FIRST SERVED ALGORITHM**

print("FIRST COME FIRST SERVE SCHEDULLING")

n = int(input("Enter number of processes: "))

p = []

at=input("Enter the Arrival Time for Each Process with Space ").split()

at = list(map(int, at))

bt=input("Enter the Burst Time for Each Process with Space ").split()

bt = list(map(int, bt))

//Sorting arrival time

p=list(zip(at,bt))

CT = []

TAT = []

WT = []

for i in range(n):

if i == 0:

CT.append(p[i][1]+p[i][0])

else:

CT.append(CT[i-1] + p[i][1])

TAT.append(CT[i] - p[i][0])

WT.append(TAT[i] - p[i][1])

Avg\_TAT = sum(WT) / n

Avg\_WT = sum(TAT) / n

print(f"\nPID\t\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time")

for i in range(n):

print(f"P{i+1}\t\t{p[i][0]}\t\t{p[i][1]}\t\t{CT[i]}\t\t{TAT[i]}\t\t{WT[i]}")

print(f"\nAverage Waiting Time: {Avg\_WT:.2f}")

print(f"Average Turnaround Time: {Avg\_TAT:.2f}")

**OUTPUT**

# EX. NO: 3 FIRST COME FIRST SERVED ALGORITHM

FIRST COME FIRST SERVE SCHEDULLING

Enter number of processes: 5

Enter the Arrival Time for Each Process with Space 3 1 4 0 2

Enter the Burst Time for Each Process with Space 1 4 2 6 3

PID Arrival Time Burst Time Completion Time Turnaround Time Waiting Time

P1 3 1 4 1 0

P2 1 4 8 7 3

P3 4 2 10 6 4

P4 0 6 16 16 10

P5 2 3 19 17 14

Average Waiting Time: 9.40

Average Turnaround Time: 6.20

**EX. NO:4(A) ROUNDROBIN ALGORITHM**

def round\_robin(n, arrival\_time, burst\_time, time\_slot):

wait\_time = 0

ta\_time = 0

temp\_burst\_time = burst\_time.copy()

x = n

CT = 0

counter = 0

i = 0

print("PID\tBT\tCT\tTAT\tWAT")

while x != 0:

if temp\_burst\_time[i] <= time\_slot and temp\_burst\_time[i] > 0:

CT += temp\_burst\_time[i]

temp\_burst\_time[i] = 0

counter = 1

elif temp\_burst\_time[i] > 0:

temp\_burst\_time[i] -= time\_slot

CT += time\_slot

if temp\_burst\_time[i] == 0 and counter == 1:

x -= 1

print(f"P{i+1}\t {burst\_time[i]}\t{CT}\t {CT-arrival\_time[i]}\t{CT-arrival\_time[i]-burst\_time[i]}")

wait\_time += CT - arrival\_time[i] - burst\_time[i]

ta\_time += CT - arrival\_time[i]

counter = 0

if i == n - 1:

i = 0

elif arrival\_time[i + 1] <= CT:

i += 1

else:

i = 0

average\_wait\_time = wait\_time \* 1.0 / n

average\_turnaround\_time = ta\_time \* 1.0 / n

print(f"\nAverage Waiting Time: {average\_wait\_time}")

print(f"Avg Turnaround Time: {average\_turnaround\_time}")

print("\n\nROUND-ROBIN SCHEDULING")

n = int(input("Enter Number of Processes: "))

arrival\_time = []

burst\_time = []

arrival\_time = [0] \* n

burst\_time = [0] \* n

pid = ["P" + str(i+1) for i in range(n)]

print("AT BT ")

for i in range(n):

arrival\_time[i],burst\_time[i]=map(int, input().split())

time\_slot = int(input("Enter Time Slot: "))

round\_robin(n, arrival\_time, burst\_time, time\_slot)

**OUTPUT**

**EX. NO:4 (A) ROUND ROBIN ALGORITHM**

ROUND-ROBIN SCHEDULING

Enter Number of Processes: 5

AT BT

0 5

1 3

2 1

3 2

4 3

Enter Time Slot: 2

PID BT CT TAT WAT

P3 1 5 3 2

P4 2 7 4 2

P2 3 12 11 8

P5 3 13 9 6

P1 5 14 14 9

Average Waiting Time: 5.4

Avg Turnaround Time: 8.2

**EX. NO:4 (B) PRIORITY SCHEDULING ALGORITHM**

print("\n\nPRIORITY NON-PRE-EMPTIVE SCHEDULING")

bt = []

pri = []

at = []

pid = []

n = 0

print("Enter the number of Processes for Scheduling: ", end="")

n = int(input())

bt = [0] \* n

pri = [0] \* n

at = [0] \* n

pid = ["P" + str(i+1) for i in range(n)]

print("AT BT PRI")

for i in range(n):

at[i],bt[i], pri[i]=map(int, input().split())

CAT = [0] \* n

waitingTime = [0] \* n

turnAroundTime = [0] \* n

bt = bt.copy()

at = at.copy()

prt = pri.copy()

pid = pid.copy()

#sorting based on arrival time and priority

for i in range(n):

for j in range(n - i - 1):

# Sorting according to pri when arrival timings are the same

if (at[j]!=0 and pri[j] > pri[j + 1]):

# Swapping pri

pri[j], pri[j + 1] = pri[j + 1], pri[j]

# Swapping burst time

bt[j], bt[j + 1] = bt[j + 1], bt[j]

# Swapping process identity

pid[j], pid[j + 1] = pid[j + 1], pid[j]

CAT[0] = at[0] + bt[0]

turnAroundTime[0] = CAT[0] - at[0]

waitingTime[0] = turnAroundTime[0] - bt[0]

for i in range(1, n):

CAT[i] = bt[i] + CAT[i - 1]

turnAroundTime[i] = CAT[i] - at[i]

waitingTime[i] = turnAroundTime[i] - bt[i]

Avg\_TAT = sum(waitingTime) / n

Avg\_WT = sum(turnAroundTime) / n

print("Priority Scheduling Algorithm:")

print(f"{'PId'}\t{'AT'}\t{'BT'}\t{'PRI'}\t{'CAT'}\t{'WAT'}\t{'TAT'}")

for i in range(n):

print(f"{pid[i]}\t{at[i]}\t{bt[i]}\t{prt[i]}\t{CAT[i]}\t{waitingTime[i]}\t{turnAroundTime[i]}")

print(f"Avg turnaround time is: {Avg\_TAT:.2f}")

print(f"Avg waiting time is: {Avg\_WT:.2f}")

**OUTPUT**

# EX. NO:4 (B) PRIORITY SCHEDULING ALGORITHM

PRIORITY NON-PRE-EMPTIVE SCHEDULING

Enter the number of Processes for Scheduling: 7

AT BT PRI

0 3 2

2 5 6

1 4 3

4 2 5

6 9 7

5 4 4

7 10 10

PId AT BT PRI CAT WAT TAT

P1 0 3 2 3 0 3

P3 2 4 6 7 1 5

P6 1 4 3 11 6 10

P4 4 2 5 13 7 9

P2 6 5 7 18 7 12

P5 5 9 4 27 13 22

P7 7 10 10 37 20 30

Avg turnaround time is: 7.71

Avg waiting time is: 13.00

**EX. NO:5 READER /WRITER PROBLEM USING SEMAPHORE**

import threading as thread

import random

#Shared Data

global x

x = 0

lock = thread.Lock() #Lock for synchronising access

def Reader():

global x

print('Reader is Reading!')

#Acquire the lock before Reading (mutex approach)

lock.acquire()

print('Shared Data:', x)

#Release the lock after Reading

lock.release()

print()

def Writer():

global x

print('Writer is Writing!')

#Acquire the lock before Writing

lock.acquire()

#Write on the shared memory

x += 1

print('Writer is Releasing the lock!')

#Release the lock after Writing

lock.release()

print()

if \_\_name\_\_ == '\_\_main\_\_':

for i in range(0, 10):

randomNumber = random.randint(0, 100) #Generate a Random number between 0 to 100

if(randomNumber > 50):

Thread1 = thread.Thread(target = Reader)

Thread1.start()

else:

Thread2 = thread.Thread(target = Writer)

Thread2.start()

Thread1.join()

Thread2.join()

**OUTPUT**

# EX. NO:5 READER /WRITER PROBLEM USING SEMAPHORE

Reader is Reading!Reader is Reading!Writer is Writing!Writer is Writing!Reader is Reading!Writer is Writing!Writer is Writing!Reader is Reading!Reader is Reading!

Reader is Reading!

Shared Data:

0

Shared Data: 0

Writer is Releasing the lock!

Writer is Releasing the lock!

Shared Data: 2

Writer is Releasing the lock!

Writer is Releasing the lock!

Shared Data: 4

Shared Data: 4

Shared Data:

4

**EX. NO:6 BANKER'S ALGORITHM FOR DEADLOCK AVOIDANCE**

'''

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project.

NumPy stands for Numerical Python.

'''

import numpy as np

def check(i):

for j in range(no\_r):

if(needed[i][j]>available[j]):

return 0

return 1

no\_p = 5

no\_r = 4

Sequence = np.zeros((no\_p,),dtype=int)

visited = np.zeros((no\_p,),dtype=int)

allocated = np.array([[4,0,0,1],[1,1,0,0],[1,2,5,4],[0,6,3,3],[0,2,1,2]])

maximum = np.array([[6,0,1,2],[1,7,5,0],[2,3,5,6],[1,6,5,3],[1,6,5,6]])

needed = maximum - allocated

print("\nNeeded Resource:\n",needed)

available = np.array([3,2,1,1])

count = 0

while( count < no\_p ):

temp=0

for i in range( no\_p ):

if( visited[i] == 0 ):

if(check(i)):

Sequence[count]=i;

count+=1

visited[i]=1

temp=1

for j in range(no\_r):

available[j] += allocated[i][j]

if(temp == 0):

break

print("The system is Safe")

print("Safe Sequence: ",Sequence)

print("Available Resource:",available)

**OUTPUT**

# EX. NO:6 BANKER'S ALGORITHM FOR DEADLOCK AVOIDANCE

Needed Resource:

[[2 0 1 1]

[0 6 5 0]

[1 1 0 2]

[1 0 2 0]

[1 4 4 4]]

The system is Safe

Safe Sequence: [0 2 3 4 1]

Available Resource: [ 9 13 10 11]

**EX. NO:7 FIRST IN FIRST OUT ALGORITHM**

# Function to find page faults using FIFO

def pageFaults(incomingStream, n, frames):

print("Incoming \t pages")

# Using Hashset to quickly check if a given

# incoming stream item in set or not

s = set()

# Queue created to store pages in FIFO manner

# since set will not store order or entry

# we will use queue to note order of entry of incoming page

queue = Queue()

page\_faults = 0

for i in range(n):

# if set has lesser item than frames

# i.e. set can hold more items

if len(s) < frames:

# If incoming item is not present, add to set

if incomingStream[i] not in s:

s.add(incomingStream[i])

# increment page fault

page\_faults += 1

# Push the incoming page into the queue

queue.put(incomingStream[i])

# If the set is full then we need to do page replacement

# in FIFO manner that is remove first item from both

# set and queue then insert incoming page

else:

# If incoming item is not present

if incomingStream[i] not in s:

# remove the first page from the queue

val = queue.queue[0]

queue.get()

# Remove from set

s.remove(val)

# insert incoming page to set

s.add(incomingStream[i])

# push incoming page to queue

queue.put(incomingStream[i])

# Increment page faults

page\_faults += 1

print(incomingStream[i], end="\t\t")

for q\_item in queue.queue:

print(q\_item, end="\t")

print()

return page\_faults

# Driver code

incomingStream = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1]

n = len(incomingStream)

frames = 3

page\_faults = pageFaults(incomingStream, n, frames)

hits = n - page\_faults

print("\nPage Faults: " + str(page\_faults))

print("Hit: " + str(hits))

**OUTPUT**

# EX. NO:7 FIRST IN FIRST OUT ALGORITHM

Incoming pages

7 7

0 7 0

1 7 0 1

2 0 1 2

0 0 1 2

3 1 2 3

0 2 3 0

4 3 0 4

2 0 4 2

3 4 2 3

0 2 3 0

3 2 3 0

2 2 3 0

1 3 0 1

Page Faults: 11

Hit: 3

**EX. NO:8 LEAST RECENTLY USED ALGORITHM**

print("Enter the numbers of frames: ",end="") capacity=int(input())

f,st,fault,pf=[],[],0,'no'

print("Enter the reference string: ",end="") s=list(map(int,input().strip().split())) print("\nString|Frame \t",end="")

for i in range(capacity): print(i,end='')

print("fault\n \n") for i in s:

if i not in f:

if len(f)<capacity: f.append(i) st.append(len(f)-1)

else:

ind=st.pop(0) f[ind]=i st.append(ind)

pf='yes' fault+=1

else:

st.append(st.pop(st.index(f.index(i)))) pf='no'

print(" %d\t\t"%i,end=' ') for x in f:

print(x,end=' ')

for x in range(capacity-len(f)): print (' ',end=' ')

print("%s"%pf)

print("\nTotal request:%d\nTotal Page Faults:%d\nFault Rate:%0.2f%%"%(len(s),fault,(fault/len(s))\*100))

**OUTPUT**

# EX. NO: 8 LEAST RECENTLY USED ALGORITHM

Enter the numbers of Frames: 3

Enter the reference String: 3 2 1 0 3 2 4 3 2 1 0 4

Capacity

012

String Frame Fault

3 3 Yes

2 3 2 Yes

1 3 2 1 Yes

0 0 2 1 Yes

3 0 3 1 Yes

2 0 3 2 Yes

4 4 3 2 Yes

3 4 3 2 No

2 4 3 2 No

1 1 3 2 Yes

0 1 0 2 Yes

4 1 0 4 Yes

Total request:12

Total Page Faults:10

FaultRate:83.33%

**EX. NO: 9(A) FIRST FIT ALGORITHM**

def firstFit(blockSize, m, processSize, n): allocation = [-1] \* n

for i in range(n): for j in range(m):

if blockSize[j] >= processSize[i]: allocation[i] = j

blockSize[j] -= processSize[i] break

print(" Process No. Process Size Block no.") for i in range(n):

print(" ", i + 1, " ", processSize[i], " ", end = " ")

if allocation[i] != -1: print(allocation[i] + 1)

else:

print("Not Allocated")

if name == ' main ':

blockSize = [100, 500, 200, 300, 600]

processSize = [212, 417, 112, 426] m = len(blockSize)

n = len(processSize) firstFit(blockSize, m, processSize, n)

**OUTPUT**

# EX. NO: 9(A) FIRST FIT ALGORITHM

Process No. Process Size Block no.

|  |  |  |
| --- | --- | --- |
| 1 | 212 | 2 |
| 2 | 417 | 5 |
| 3 | 112 | 2 |
| 4 | 426 | Not Allocated |

**EX. NO: 9(B) BEST FIT ALGORITHM**

def bestFit(blockSize, m, processSize, n): allocation = [-1] \* n

for i in range(n): bestIdx = -1

for j in range(m):

if blockSize[j] >= processSize[i]: if bestIdx == -1:

bestIdx = j

elif blockSize[bestIdx] > blockSize[j]: bestIdx = j

if bestIdx != -1: allocation[i] = bestIdx

blockSize[bestIdx] -= processSize[i]

print("Process No. Process Size Block no.") for i in range(n):

print(i + 1, " ", processSize[i],

end = " ") if allocation[i] != -1:

print(allocation[i] + 1) else:

print("Not Allocated")

if name == ' main ':

blockSize = [100, 500, 200, 300, 600]

processSize = [212, 417, 112, 426] m = len(blockSize)

n = len(processSize) bestFit(blockSize, m, processSize, n)

**OUTPUT**

# EX. NO: 9(B) BEST FIT ALGORITHM

Process No. Process Size Block no.

|  |  |  |
| --- | --- | --- |
| 1 | 212 | 4 |
| 2 | 417 | 2 |
| 3 | 112 | 3 |
| 4 | 426 | 5 |

**EX. NO: 9(C) WORST FIT ALGORITHM**

def worstFit(blockSize, m, processSize, n): allocation = [-1] \* n

for i in range(n): wstIdx = -1

for j in range(m):

if blockSize[j] >= processSize[i]: if wstIdx == -1:

wstIdx = j

elif blockSize[wstIdx] < blockSize[j]: wstIdx = j

if wstIdx != -1:

allocation[i] = wstIdx blockSize[wstIdx] -= processSize[i]

print("Process No. Process Size Block no.") for i in range(n):

print(i + 1, " ", processSize[i], end = " ")

if allocation[i] != -1: print(allocation[i] + 1)

else:

print("Not Allocated")

if name == ' main ':

blockSize = [500, 100, 200, 300, 600]

processSize = [212, 417, 112, 426] m = len(blockSize)

n = len(processSize) worstFit(blockSize, m, processSize, n)

**OUTPUT**

# EX. NO: 9(C) WORST FIT ALGORITHM

Process No. Process Size Block no.

|  |  |  |
| --- | --- | --- |
| 1 | 212 | 5 |
| 2 | 417 | 1 |
| 3 | 112 | 5 |
| 4 | 426 | Not Allocated |

**EX. NO: 10 INTER - PROCESS COMMUNICATION**

# Example of using shared memory with strings

from multiprocessing.shared\_memory import SharedMemory

from multiprocessing import Process

# task executed in a child process

def task(shared\_mem):

# write some string data to the shared memory

shared\_mem.buf[:24] = b'Hello from child process'

# close as no longer needed

shared\_mem.close()

# protect the entry point

if \_\_name\_\_ == '\_\_main\_\_':

# create a shared memory

shared\_mem = SharedMemory(create=True, size=100)

# create a child process

process = Process(target=task, args=(shared\_mem,))

# start the child process

process.start()

# wait for the child process to finish

process.join()

# report the shared memory

data = bytes(shared\_mem.buf[:24]).decode()

print(data)

# close the shared memory

shared\_mem.close()

# release the shared memory

shared\_mem.unlink()

**OUTPUT**

# EX. NO: 10 INTER - PROCESS COMMUNICATION

# Hello from child process