#1

#!/bin/bash

# Program to print sum of first N numbers

echo "Enter a number: "

read N

if [ $N -le 0 ]

then

    echo "Non-positive number detected, terminating..."

else

    i=$N

    sum=0

    while [ $i -gt 0 ]

    do

        sum=$[$sum+$i]

        i=$[$i-1]

    done

    echo "Sum of first $N numbers is $sum"

fi

OUTPUT

Enter a number:

12

Sum of first 12 numbers is 78

Enter a number:

-4

Non-positive number detected, terminating...

#2

#!/bin/bash

# Program to print fibonacci series

read -p "Enter number of terms of Fibonacci series to be printed : " N

if [ $N -le 0 ]

then

    echo "Non-positive limit detected, terminating...."

else

    i=$N

    first=0

    second=1

    temp=0

    echo -n "$first "

    while [ $i -gt 1 ]

    do

        echo -n "$second "

        temp=$second

        second=$[$first+$second]

        first=$temp

        i=$[$i-1]

    done

    echo ""

fi

OUTPUT

Enter number of terms of Fibonacci series to be printed : 12

0 1 1 2 3 5 8 13 21 34 55 89

Enter number of terms of Fibonacci series to be printed : -3

Non-positive limit detected, terminating....

#3

#!/bin/bash

# Program to calculate factorial

read -p "Enter number whose factorial is to be calculated :: " f

N=$f

fact=1

if [ $f -eq 0 ]

then

    echo "Factorial of 0 is 1"

else

    if [ $f -lt 0 ]

    then

        echo "Factorial of negative numbers is too complicated."

    else

        while [ $N -ge 1 ]

        do

            fact=$[$fact\*$N]

            N=$[$N-1]

        done

        echo "Factorial of $f is $fact."

    fi

fi

OUTPUT

Enter number whose factorial is to be calculated :: 10

Factorial of 10 is 3628800.

Enter number whose factorial is to be calculated :: -3

Factorial of negative numbers is too complicated.

Enter number whose factorial is to be calculated :: 0

Factorial of 0 is 1

#4

#!/bin/bash

# Program to find if string is palindrome or not

read -p "Enter a string :: " s

len=${#s}

y=$[$len-1]

reversed=""

for ((i=0;i<$len;i++)) do

    reversed="$reversed${s:$y:1}"

    y=$[$y-1]

done

if [ $s = $reversed ]

then

    echo "$s is a palindrome."

else

    echo "$s is not a palindrome."

fi

OUTPUT

Enter a string :: advaith

advaith is not a palindrome.

Enter a string :: cittic

cittic is a palindrome.

#5

# Program to implement First Come First Served CPU scheduling algorithm

print("First Come First Served scheduling Algorithm")

print("============================================\n")

headers = ['Processes','Arrival Time','Burst Time','Waiting Time'

            ,'Turn-Around Time','Completion Time']

# Dictionary to store the output

out = dict()

# Get number of processes from User

N = int(input("Number of processes : "))

a, b = 0, 0

# Get Arrival time and Burst time of N processes

for i in range(0,N):

    k = f"P{i+1}"

    a = int(input(f"Enter Arrival time of process{i+1} :: "))

    b = int(input(f"Enter Burst time of process{i+1} :: "))

    out[k] = [a,b]

# storing processes in order of increasing arrival time

out = sorted(out.items(),key=lambda i:i[1][0])

# storing Completion times

for i in range(0,N):

    if i == 0:

        out[i][1].append(out[i][1][0]+out[i][1][1])

    else:

        out[i][1].append(out[i-1][1][2]+out[i][1][1])

# storing turn-around times

for i in range(0,N):

    out[i][1].append(out[i][1][2]-out[i][1][0])

# storing waiting time

for i in range(0,N):

    out[i][1].append(out[i][1][3]-out[i][1][1])

# storing avg waiting time and avg turn around time

avgWaitTime = 0

avgTATime = 0

for i in range(0,N):

    avgWaitTime += out[i][1][4]

    avgTATime += out[i][1][3]

avgWaitTime /= N

avgTATime /= N

print(f"\n{headers[0]:^15}{headers[1]:^15}{headers[2]:^15}{headers[3]:^15}{headers[4]:^20}{headers[5]:^20}")

for a in out:

    print(f"{a[0]:^15}{a[1][0]:^15}{a[1][1]:^15}{a[1][4]:^15}{a[1][3]:^20}{a[1][2]:^20}")

print(f"\nAverage Waiting Time : {avgWaitTime:.2f}\nAverage Turn-Around Time : {avgTATime:.2f}")

OUTPUT

First Come First Served scheduling Algorithm

============================================

Number of processes : 3

Enter Arrival time of process1 :: 0

Enter Burst time of process1 :: 5

Enter Arrival time of process2 :: 3

Enter Burst time of process2 :: 9

Enter Arrival time of process3 :: 6

Enter Burst time of process3 :: 6

Processes Arrival Time Burst Time Waiting Time Turn-Around Time Completion Time

P1 0 5 0 5 5

P2 3 9 2 11 14

P3 6 6 8 14 20

Average Waiting Time : 3.33

Average Turn-Around Time : 10.00

#6

# Program to implement Shortest Job First CPU scheduling algorithm

print("Shortest Job First Algorithm")

print("============================\n")

headers = ['Processes','Arrival Time','Burst Time','Waiting Time'

            ,'Turn-Around Time','Completion Time']

# Dictionary to store the output

out = dict()

# Get number of processes from User

N = int(input("Number of processes : "))

a, b = 0, 0

# Get Arrival time and Burst time of N processes

for i in range(0,N):

    k = f"P{i+1}"

    a = int(input(f"Enter Arrival time of process{i+1} :: "))

    b = int(input(f"Enter Burst time of process{i+1} :: "))

    out[k] = [a,b]

# storing processes in order of increasing arrival time

out = sorted(out.items(),key=lambda i:i[1][0])

readyQ = [x for x in out[1:]]

# Storing ready queue in order of increasing Burst Times

readyQ.sort(key=lambda i:i[1][1])

# Setting completion time of first process as its burst time

out[0][1].append(out[0][1][1])

# Calculating Completion times of rest of the processes

for i in range(0,len(readyQ)):

    if i == 0:

        readyQ[i][1].append(readyQ[i][1][1] + out[0][1][2])

    else:

        readyQ[i][1].append(readyQ[i][1][1] + readyQ[i-1][1][2])

out.sort(key=lambda i:i[0])

# Storing turn around times

for i in range(0,N):

    out[i][1].append(out[i][1][2]-out[i][1][0])

# Storing waiting times

for i in range(0,N):

    out[i][1].append(out[i][1][3]-out[i][1][1])

# storing avg waiting time and avg turn around time

avgWaitTime = 0

avgTATime = 0

for i in range(0,N):

    avgWaitTime += out[i][1][4]

    avgTATime += out[i][1][3]

avgWaitTime /= N

avgTATime /= N

print(f"\n{headers[0]:^15}{headers[1]:^15}{headers[2]:^15}{headers[3]:^15}{headers[4]:^20}{headers[5]:^20}")

for a in out:

    print(f"{a[0]:^15}{a[1][0]:^15}{a[1][1]:^15}{a[1][4]:^15}{a[1][3]:^20}{a[1][2]:^20}")

print(f"\nAverage Waiting Time : {avgWaitTime:.2f}\nAverage Turn-Around Time : {avgTATime:.2f}")

OUTPUT

Shortest Job First Algorithm

============================

Number of processes : 4

Enter Arrival time of process1 :: 0

Enter Burst time of process1 :: 7

Enter Arrival time of process2 :: 2

Enter Burst time of process2 :: 4

Enter Arrival time of process3 :: 4

Enter Burst time of process3 :: 1

Enter Arrival time of process4 :: 5

Enter Burst time of process4 :: 5

Processes Arrival Time Burst Time Waiting Time Turn-Around Time Completion Time

P1 0 7 0 7 7

P2 2 4 6 10 12

P3 4 1 3 4 8

P4 5 5 7 12 17

Average Waiting Time : 4.00

Average Turn-Around Time : 8.25

#8

# Parent sends n numbers to child

import os

r, w = os.pipe()

pid = os.fork()

if pid:

    os.close(r)

    w = os.fdopen(w,'w')

    x = []

    n = int(input("Enter value of n : "))

    print(f"Enter {n} numbers :")

    for i in range(0,n):

        x.append(int(input(f"Entry #{i+1} : ")))

    w.write(str(x))

    print(f"Parent wrote : {x}")

    w.close()

else:

    os.close(w)

    r = os.fdopen(r)

    y = r.read()

    y = y.split(", ")

    print(f"Child read :")

    for z in y:

        s = ''

        for i in z:

            if i not in '[]':

                s += i

        print(int(s))

    r.close()

OUPUT

Enter value of n : 4

Enter 4 numbers :

Entry #1 : 23

Entry #2 : 34

Entry #3 : 43

Entry #4 : 123

Parent wrote : [23, 34, 43, 123]

Child read :

23

34

43

123

#9

# Parent sends numbers to child, child returns Prime numbers

import os

import math

import numpy as np

r, w = os.pipe()

r1, w1 = os.pipe()

pid = os.fork()

def isprime(num):

    a=2

    while a<=math.sqrt(num):

        if num%a<1:

            return False

        a=a+1

    return num>1

if pid:

    os.close(r)

    os.close(w1)

    w = os.fdopen(w, 'w')

    n = int(input("Parent >> Enter value of n : "))

    x = []

    print(f"Parent >> Enter {n} numbers")

    for i in range(0,n):

        x.append(int(input(f"#{i+1} : ")))

    z = np.array(x)

    y = str(z)

    w.write(y)

    print("Parent >> Sent the numbers to child.")

    w.close()

    r1 = os.fdopen(r1)

    s = r1.read()

    if s:

        print("Parent >> Prime numbers are : ")

        print(s)

    r1.close()

else :

    os.close(w)

    os.close(r1)

    r = os.fdopen(r)

    nums = r.read()

    d = ''

    for i in nums:

        for j in i:

            if j not in '[]':

                d += j

    arr = []

    for z in d.split(' '):

        if z != '':

            if isprime(int(z)):

                arr.append(int(z))

    r.close()

    w1 = os.fdopen(w1,'w')

    primes = str(arr)

    if primes != '[]':

        w1.write(primes)

        print('Child  >> Sent Primes to parent.')

    else:

        print('Child  >> No prime numbers to send.')

    w1.close()

OUPUT

Parent >> Enter value of n : 4

Parent >> Enter 4 numbers

#1 : 12

#2 : 23

#3 : 31

#4 : 44

Parent >> Sent the numbers to child.

Child >> Sent Primes to parent.

Parent >> Prime numbers are :

[23, 31]

Parent >> Enter value of n : 3

Parent >> Enter 3 numbers

#1 : 12

#2 : 22

#3 : 42

Parent >> Sent the numbers to child.

Child >> No prime numbers to send.

#10

# Parent sends string to child, child says if it is palindrome or not

import os

def palindrome(text):

    return text == text[::-1]

r, w = os.pipe()

r1, w1 = os.pipe()

pid = os.fork()

if pid:

    os.close(r)

    os.close(w1)

    w = os.fdopen(w, 'w')

    string = str(input("Parent >> Enter a string : "))

    w.write(string)

    print("Parent >> String sent to child.")

    w.close()

    r1 = os.fdopen(r1)

    s = r1.read()

    if s == 'True':

        print("Parent >> It is a palindrome.")

    else:

        print("Parent >> It is not a palindrome.")

    r1.close()

else:

    os.close(w)

    os.close(r1)

    r = os.fdopen(r)

    s = r.read()

    print("Child  >> Received string from parent.")

    r.close()

    w1 = os.fdopen(w1,'w')

    w1.write(str(palindrome(s)))

    print("Child  >> Sent result to parent.")

    w1.close()

OUTPUT

Parent >> Enter a string : Advaith

Parent >> String sent to child.

Child >> Received string from parent.

Child >> Sent result to parent.

Parent >> It is not a palindrome.

Parent >> Enter a string : cittic

Parent >> String sent to child.

Child >> Received string from parent.

Child >> Sent result to parent.

Parent >> It is a palindrome.

#11