UNIVERSIDAD DE SAN BUENAVENTURA

JEFFERSON ANDREY DAZA CARO

TALLER #2

|  |  |  |
| --- | --- | --- |
| Algoritmo cualitativo | Se desea saber la receta para hacer un pollo frito | 1. Colocar el pollo durante 20 minutos en agua a 90 grados 2. Se coloca el pollo en aceite y se deja fritar por 15 minutos 3. Se agrega las salsas al gusto |
| Algoritmo  cuantitativo | Se desea saber la multiplicación de dos números | print("1. declarar las variables",  "\n", "Ingrese los valores que desea sumar")  a=input()  a=int(a)  b=input()  b=int(b)  print("2. Se declara otra variable que almacenara el resultado de la suma:")  c=a\*b  print("3. Se muestra el resultado de la operacion: ", c) |
| Algoritmo computacional | Se desea calcular el área de un cuadrado | base=int(input(“Ingrese la base”))  altura=int(input(“Ingrese la alura”))  area=base\*altura  print(“El área es: ”,rea) |
| Algoritmo no computacional | Algoritmo para comprar cosas | 1. Ir a una tienda 2. Seleccionar los productos deseados 3. Sacar la billetera 4. Ir a la caja 5. Pagarle al cajero |
| Algoritmo de Ordenamiento | Algoritmo que organice n cantidad de números de menor a mayor | print("inresa la cantidad de numeros que seran ingresados: ") n=input() n=int(n)  A=[0 for i in range(n)]  for i in range(n):  A[i]=int(input("Ingrese un numero: "))  #Metodo Burbuja  for i in range(n-1):  for j in range(n-1):  if A[j]>A[j+1]: # si numeroActual > numeroSiguiente  aux=A[j]  A[j]=A[j+1]  A[j+1]=aux  print("Los numero de mayor a menor son:") for i in range(n):  print (A[i]) |
| Algoritmo de búsqueda | al contrario de realizar operaciones o secuenciar elementos, se dedica a encontrar dentro de una lista que ingresa, uno o varios elementos en particular que cumplan con el conjunto de condiciones dadas.  Se desea encontrar un elemento dentro de una lista de elementos. | package algoritmo\_busqueda;  import java.util.ArrayList;  import javax.swing.JOptionPane;  public class Algoritmo\_busqueda {  public static void main(String[] args) {  Object dato;  ArrayList lista = new ArrayList();  boolean enc = false;  lista.add("perro");  lista.add("gato");  lista.add("12");  lista.add("10");  dato = JOptionPane.showInputDialog("Ingrese el objecto que quiere buscar ");  int i = 0;  while (i < lista.size() && enc == false) {  if (lista.get(i).equals(dato)) {  enc = true;  }  i++;  }  if (enc == false) {  JOptionPane.showMessageDialog(null, "El objeto no esta en la lista");  } else {  JOptionPane.showMessageDialog(null, "El objeto si esta dentro de la lista");  }  }  } |
| Algoritmo de encaminamiento | Realizar un algoritmo que lea el siguiente grafo con un recorrido a lo ancho.  C:\Users\USUARIO\Documents\Universidad\QUINTO SEMESTRE\ANALISIS Y DISEÑO DE ALGORITMOS\PRIMER CORTE\Untitled Diagram.jpg  El recorrido debe mostrarse de la siguiente manera:  A B C D F E G H | from collections import deque   class Grafo(object):  def \_\_init\_\_(self):  self.relaciones = {}   def \_\_str\_\_(self):  return str(self.relaciones)   def agregar(grafo, elemento):  grafo.relaciones.update({elemento: []})   def relacionar(grafo, elemento1, elemento2):  relacionDireccional(grafo, elemento1, elemento2)  relacionDireccional(grafo, elemento2, elemento1)   def relacionDireccional(grafo, origen, destino):  grafo.relaciones[origen].append(destino)  a = "A" b = "B" c = "C" d = "D" e = "E" f = "F" g = "G" h = "H"    grafo = Grafo() agregar(grafo, a) agregar(grafo, c) agregar(grafo, b) agregar(grafo, f) agregar(grafo, h) agregar(grafo, g) agregar(grafo, d) agregar(grafo, e)  relacionar(grafo, a, b) relacionar(grafo, a, c) relacionar(grafo, a, d) relacionar(grafo, a, c) relacionar(grafo, c, e) relacionar(grafo, c, g) relacionar(grafo, c, d) relacionar(grafo, g, h) relacionar(grafo, g, f) relacionar(grafo, f, b)  def lecturaAncho(grafo, elementoInicial, funcion, cola = deque(), elementosRecorridos = []):  if not elementoInicial in elementosRecorridos:  funcion(elementoInicial)  elementosRecorridos.append(elementoInicial)  if(len(grafo.relaciones[elementoInicial]) > 0):  cola.extend(grafo.relaciones[elementoInicial])  if len(cola) != 0 :  lecturaAncho(grafo, cola.popleft(), funcion, cola, elementosRecorridos)  def imprimir (elemento):  print (elemento)  lecturaAncho(grafo, a, imprimir) |
| Algoritmo de marcaje | Este algoritmo utiliza la automatización para fijar los precios de forma dinámica, basándose en factores como el comportamiento del cliente.  Se desea saber el costo de un viaje en uber dependiendo el numero de usuarios utilizando la aplicación. | package algoritmo\_marcaje;  import javax.swing.JOptionPane;  public class Algoritmo\_marcaje {  public static void main(String[] args) {  int cant = 0;  cant = Integer.parseInt(JOptionPane.showInputDialog("Ingrese el numero de usuarios conectados "));  if (cant <= 2000) {  JOptionPane.showMessageDialog(null, "Tarida normal");  }  if (2000 < cant && cant <= 4000) {  JOptionPane.showMessageDialog(null, "Tarida aumentada un 25%");  }  if (cant > 4000) {  JOptionPane.showMessageDialog(null, "Tarida aumentada un 50%%");  }  }  } |
| Algoritmo numérico | **Dados un determinado número de días, calcular cuantos segundos hay en dicho número.** | nd=int(input("Ingrese el numero de días que desea transformar a segundos:")) s=nd\*86400 print(nd," días son ",s," segundos ") |
| Algoritmo las Vegas | estos ofrecen una respuesta correcta, y si existe algún fallo en los cálculo lo informa. se caracterizan porque el resultado siempre será correcto, pero el sistema puede utilizar más de los recursos anticipados o más tiempo del estimado.  Se desea saber que turno tienen un grupo de personas para acceder a un hospital que tiene como máximo 6 cupos simultaneos | package algoritmo\_las\_vegas;  import java.util.Random;  import javax.swing.JOptionPane;  public class Algoritmo\_las\_vegas {  public static void main(String[] args) {  //int a = (int)Math.random();  Random ale = new Random();  int a = ale.nextInt(9);  if (a <= 6) {  JOptionPane.showMessageDialog(null, "Correcto, su turno es el: " + a);  } else {  JOptionPane.showMessageDialog(null, "ERROR!!!");  }  }  } |
| Algoritmo Montecarlo | calcular el área de un círculo de radio 1; lo que es lo mismo a decir que aproximemos el valor de [π](https://relopezbriega.github.io/blog/2015/03/14/el-dia-pi/). | import inline as inline import matplotlib as matplotlib import matplotlib.pyplot as plt import numpy as np # importando numpy import pandas as pd # importando pandas from scipy import stats  np.random.seed(1984) # para poder replicar el random  matplotlib  def mc\_pi\_aprox(N=10000):  plt.figure(figsize=(4,4)) # tamaño de la figura  x, y = np.random.uniform(-1, 1, size=(2, N))  interior = (x\*\*2 + y\*\*2) <= 1  pi = interior.sum() \* 4 / N  error = abs((pi - np.pi) / pi) \* 100  exterior = np.invert(interior)  plt.plot(x[interior], y[interior], 'b.')  plt.plot(x[exterior], y[exterior], 'r.')  plt.plot(0, 0, label='$\hat \pi$ = {:4.4f}\nerror = {:4.4f}%'  .format(pi,error), alpha=0)  plt.axis('square')  plt.legend(frameon=True, framealpha=0.9, fontsize=16)   print("pi= ",pi)  print("error=",error)  mc\_pi\_aprox() |
| Algoritmo cotidiano | Muchas de las decisiones que se toman desde que uno se despierta por la mañana pertenecen a este grupo. | 1. Tender la cama 2. Cepillarme 3. Hacer el desayuno 4. Desayunar 5. Bañarme 6. Cepillarme again 7. Ir a la universidad |
| Algoritmo heurístico | Encuentra el camino más corto teniendo en cuenta los siguientes parámetros:  T=jugador o punto inicial  S=Salida  . = el camino  # = Pared u obstáculo. | class Mapa:  def \_\_init\_\_(self, archivo="mapa.txt"):  self.mapa = leerMapa(archivo)  self.fil = len(self.mapa)  self.col = len(self.mapa[0])    def \_\_str\_\_(self):  salida = ""  for f in range(self.fil):  for c in range(self.col):  if self.mapa[f][c] == 0:  salida += ". "  if self.mapa[f][c] == 1:  salida += "# "  if self.mapa[f][c] == 2:  salida += "T "  if self.mapa[f][c] == 3:  salida += "S "  salida += "\n"  return salida  # ---------------------------------------------------------------------  # Funciones # ---------------------------------------------------------------------  # Quita el ultimo caracter de una lista. def quitarUltimo(lista):  for i in range(len(lista)):  lista[i] = lista[i][:-1]  return lista  # Covierte una cadena en una lista.  def listarCadena(cadena):  lista = []  for i in range(len(cadena)):  if cadena[i] == ".":  lista.append(0)  if cadena[i] == "#":  lista.append(1)  if cadena[i] == "T":  lista.append(2)  if cadena[i] == "S":  lista.append(3)  return lista  # Lee un archivo de texto y lo convierte en una lista. def leerMapa(archivo):  mapa = open(archivo, "r")  mapa = mapa.readlines()  mapa = quitarUltimo(mapa)  for i in range(len(mapa)):  mapa[i] = listarCadena(mapa[i])  return mapa  # ---------------------------------------------------------------------  def main():  mapa = Mapa()  print (mapa)  return 0  if \_\_name\_\_ == '\_\_main\_\_':  main() |
| Algoritmo de escalada | se comienza con una solución insatisfactoria (que no cumple la entrada y la salida), y se la va modificando aproximándose a lo que se busca. En algún momento, estaremos cerca de (o llegaremos a) la solución correcta.  Se desea saber si dado un numero del 0 al 15 ingresado, es el ganador para una rifa. | package algoritmo\_escalada;  import java.util.Random;  import javax.swing.JOptionPane;  public class Algoritmo\_escalada {  public static void main(String[] args) {  Random ale = new Random();  int obj = Integer.parseInt(JOptionPane.showInputDialog("que numero quiere buscar del 0 al 15?"));  boolean enc = false;  while (enc == false) {  int a = ale.nextInt(16);  if (a == obj) {  JOptionPane.showMessageDialog(null, "Termina la busqueda con exito");  enc = true;  } else {  JOptionPane.showMessageDialog(null, "Volviendo a intentarlo");  }  }  }  } |
| Algoritmo voraz | Realizar un algoritmo que determine mediante una matriz el número de monedas que serán utilizadas para alcanzar una cifra | denominaciones = [50, 100, 200, 500, 1000]  def devolverValor(cambio, denominaciones):  vueltas = [0] \* len(denominaciones)  for pos, moneda in enumerate(reversed(denominaciones)):   while moneda <= cambio:  cambio = cambio \  - moneda  vueltas[pos] += 1  return (vueltas)  print(devolverValor(300, denominaciones)) |
| Algoritmo determinista | es completamente lineal (cada paso tiene un paso sucesor y un paso predecesor) y por lo tanto predictivo, si se conocen sus entradas y su forma de proceder  Se desea conocer el resultado de un numero random elevado a un número ingresado por el usuario. | package algotirmo\_determinista;  import java.util.Random;  import javax.swing.JOptionPane;  public class Algotirmo\_determinista {  public static void main(String[] args) {  Random ran = new Random();  int a = Integer.parseInt(JOptionPane.showInputDialog("Ingrese un entero para el exponente diferente de 0"));  int b = ran.nextInt(17);  int res = 0;  if (a == 1) {  res = b;  } else {  for (int i = 1; i < a; i++) {  if (res == 0) {  res = b;  res = res \* b;  } else {  res = res \* b;  }  }  }  JOptionPane.showMessageDialog(null, "El resultado es: " + res);  }  } |
| Algoritmo no determinista | Representa el siguiente arbol por profundidad mostrando en respectivo orden los nombres de los integrantes del arbol  C:\Users\USUARIO\Documents\Universidad\QUINTO SEMESTRE\ANALISIS Y DISEÑO DE ALGORITMOS\PRIMER CORTE\ARBG.jpg  Solución:  Abuela – tia1 –tia2 –sobrino – madre – hijo1 – hijo2 – hijo3 | class Arbol:  def \_\_init\_\_(self, elemento):  self.hijos = []  self.elemento = elemento  abuela = "ABUELA" madre = "MADRE" tia1 = "TIA1" tia2 = "TIA2" hijo1 = "HIJO1" hijo2 = "HIJO2" hijo3 = "HIJO3" sobrino = "SOBRINO"  def agregarElemento(arbol, elemento, elementoPadre):  subarbol = buscarSubarbol(arbol, elementoPadre);  subarbol.hijos.append(Arbol(elemento))  def buscarSubarbol(arbol, elemento):  if arbol.elemento == elemento:  return arbol  for subarbol in arbol.hijos:  arbolBuscado = buscarSubarbol(subarbol, elemento)  if (arbolBuscado != None):  return arbolBuscado  return None  arbol = Arbol(abuela) agregarElemento(arbol, tia1, abuela) agregarElemento(arbol, tia2, abuela) agregarElemento(arbol, sobrino, tia2) agregarElemento(arbol, madre, abuela) agregarElemento(arbol, hijo1, madre) agregarElemento(arbol, hijo2, madre) agregarElemento(arbol, hijo3, madre)   def Profundidad(arbol, funcion):  funcion(arbol.elemento)  for hijo in arbol.hijos:  Profundidad(hijo, funcion)  def printElement(element):  print (element) Profundidad(arbol, printElement) |
| Algoritmo de vuelta atrás | Obtener todas las combinaciones de sumar un numero en concreto. | import java.util.ArrayList;    public class Ejercicio\_recursividad\_DDR\_20 {    public static void main(String[] args) {    int n = 5;  ArrayList<Integer> numeros = new ArrayList<>();  combinacionesSuma(n, numeros, 0);  }    public static void combinacionesSuma(int numero, ArrayList<Integer> numeros, int suma) {    //Caso base  if (suma == numero) {    //Muestro los numeros  System.out.println(numeros);    } else {  for (int i = 1; i <= numero; i++) {  suma += i;  //Si la suma es mayor que el numero no hacemos la recursividad  if (suma <= numero) {  //añado el numero  numeros.add(i);  combinacionesSuma(numero, numeros, suma);  //elimino el numero  numeros.remove(numeros.indexOf(i));  }  //deshago la suma  suma -= i;    }    }    }    } |
| Programación dinámica | Realice un algoritmo mediante la programación dinámica que le diga cuantas monedas debe utilizar para dar el cambio y la denominación de las mismas. | def vueltasProgDin(listaValoresMonedas,vueltas,minMonedas,monedasUsadas):  for pesos in range(vueltas+1):  conteoMonedas = pesos  nuevaMoneda = 1  for j in [m for m in listaValoresMonedas if m <= pesos]:  if minMonedas[pesos-j] + 1 < conteoMonedas:  conteoMonedas = minMonedas[pesos-j]+1  nuevaMoneda = j  minMonedas[pesos] = conteoMonedas  monedasUsadas[pesos] = nuevaMoneda  return minMonedas[vueltas]  def imprimirMonedas(monedasUsadas,vueltas):  moneda = vueltas  while moneda > 0:  estaMoneda = monedasUsadas[moneda]  print(estaMoneda)  moneda = moneda - estaMoneda  def main():  cantidad = 1350  listaM = [50,100,200,500,1000]  monedasUsadas = [0]\*(cantidad+1)  conteoMonedas = [0]\*(cantidad+1)   print("Dar unas vueltas de",cantidad,"pesos requiere")  print(vueltasProgDin(listaM,cantidad,conteoMonedas,monedasUsadas),"monedas")  print("De los siguientes valores")  imprimirMonedas(monedasUsadas,cantidad) main() |