TEMA 3:

Tipos de Datos Estructurados

ÍNDICE

3

Final de estos apartados:

EJEMPLOS ILUSTRATIVOS TRABAJO PERSONAL

Introducción: Tipos de datos estructurados

Además de los tipos de datos básicos ya definidos, habitualmente los lenguajes de programación permiten a los programadores definir sus propios *tipos*.

- En lenguajes estructurados hay que definirlos, mediante su declaración. La declaración comienza con la palabra reservada **TIPO**.
- Declarar tipos es indicar al compilador el conjunto de valores y las posibles operaciones a realizar sobre este tipo de dato y, por tanto, sobre la información que representa.
- Los tipos de datos que veremos son:
  - 3.1. Tipos enumerados
  - 3.2. Subrangos
  - 3.3. Vectores y matrices
  - 3.4. Cadenas de caracteres
  - 3.5. Registros Copiar módulo record

Instalar modulo numpy:
http://www.cdlibre.org/consultar/catalogo/Python_Bibliotecas.html

3.1. Tipo Enumerado

Tipo Enumerado

- En muchos programas necesitamos referirnos y utilizar objetos distintos de los tipos básicos, por ejemplo: colores, planetas, días de la semana....
- Algunos lenguajes permiten definir un tipo de datos denominado enumerado utilizando una lista de identificadores que serán sus valores. La forma de definirlo será:

identTipo = (*valores*) {los valores se separan por comas}

Ejemplos:

Tipo

tcolores = (rojo, verde, azul)

tdias = (lunes, martes, miercoles, jueves, viernes, sabado, domingo)

3.1. Tipo Enumerado

- Son **ordinales** , es decir tienen relación de orden dada por la disposición de los valores en la definición y escalares.
- Operaciones que se pueden utilizar sobre estos tipos en lenguajes estructurados :
  - **ord (valor)➔** retorna un entero que representa la posición que ocupa el valor en la definición (*).
  - **pred (valor)➔** retorna el valor anterior (da error si se trata del primero).
  - **succ(valor)➔** retorna el siguiente valor (da error si se trata del último).
  - **Operaciones relacionales** (actúan en función del orden de declaración).
  - **Asignación**
- En el caso del valor máximo de un tipo enumerado, succ no está definido, y, para su valor mínimo, no está definido pred.
- (*) El número de orden de cada elemento comienza en 0 para el primer elemento.
- Las variables de tipo enumerado sólo pueden tomar los valores especificados en su definición.

3.1. Tipo Enumerado

Tipo Enumerado. Ejemplo

```
Algoritmo Dias_Semana
  {El siguiente programa muestra los días de la semana por medio de tipos enumerados}
 TIPO
    Dia_Semana = (Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo)
 Variable
    dias :Dia_Semana
    i    : entero                    Inicio
                                          dias:=lunes;
                                          desde  i←  1 hasta  7  hacer
                                                              Según dias hacer
                                                                        Lunes     :  Escribir('Lunes    ')
                                                                        Martes    :  Escribir ('Martes   ‘)
                                                                        Miercoles : Escribir ('Miércoles‘)
                                                                        Jueves    :  Escribir ('Jueves   ‘)
                                                                        Viernes   :  Escribir ('Viernes  ')
                                                                        Sabado    : Escribir ('Sabado   ')
                                                                        Domingo   :Escribir ('Domingo  ')
                                                               fin_segun
                                                               dias:=succ(dias)
                                                    fin_desde
                                          Fin.
```

3.1.1. Tipo Enumerado PYTHON

- A partir de Python 3.4 el lenguaje implementa enumeraciones de forma nativa, por defecto, vía módulo enum: https://www.python.org/dev/peps/pep-0435/ **#Información**

```
from enum import Enum

DiasSemana = Enum ('DiasSemana','Lunes
Martes Miercoles Jueves Viernes Sabado
Domingo')

for dia in DiasSemana:
    print (dia) #Muestra toda la variable

for dia in DiasSemana:
    print (dia.name) #Muestra nombres

for dia in DiasSemana:
    print (dia.value) #Muestra valores que toma
```

```
DiasSemana.Lunes
DiasSemana.Martes
DiasSemana.Miercoles
DiasSemana.Jueves
DiasSemana.Viernes
DiasSemana.Sabado
DiasSemana.Domingo
Lunes
Martes
Miercoles
Jueves
Viernes
Sabado
Domingo
1
2
3
4
5
6
7
```

3.1.2. Tipo Enumerado características

- Un valor de un tipo enumerado no puede leerse ni escribirse directamente, por lo que habrá que estudiar las ventajas (claridad y relación de orden) e inconvenientes de su declaración.
- Suelen utilizarse en lenguajes estructurados, como variables de control en bucles *desde*, como índices de vectores y como opciones en sentencias *según..sea*.
- Se utilizan principalmente para aportar claridad al código del programa
- En Python puede compararse, pero no existen muchas de las operaciones definidas en pseudocódigo

3.2. Tipo Subrango

- El tipo subrango en lenguajes estructurados es un intervalo dentro de un cierto tipo de datos **ordinal**.
- Los subrangos solo se utilizarán cuando se conozcan los límites de los valores que tomarán las futuras variables con seguridad.
- Se suelen utilizar para el control de errores, por ejemplo: si un número está comprendido entre 1..10.
- Aportan claridad a los programas. Se suelen utilizar sobre vectores.
- Sintaxis :

  identSubrang = literal1..literal2

  por ejemplo: 0..10 (subrango de enteros).

- Donde *literal1* y *literal2* deben ser del mismo tipo, y siempre literal1<= literal2.

3.2.1. Tipo Subrango en Python

- Como se estudió en el tema 1, El tipo range(), se emplea como contador del bucle for. Internamente en Python, crea una lista inmutable de números enteros en sucesión aritmética.
- El tipo range() puede tener uno, dos o tres argumentos numéricos enteros.

Nota: En Python 2, range() se consideraba una función, pero en Python 3 no se considera una función, sino como un tipo de datos (concretamente, una lista de enteros inmutable), aunque se utiliza como si fuera una función.

- Recordemos:

  range(n) = range(0, n) = [0, 1, ... , n-1].

  Nota: Para ver los valores de la lista creada con range(), es necesario convertirla a lista mediante el tipo list().

  ```
  >>> range(5)
  range(0, 5)
  >>> list(range(5)) [0, 1, 2, 3, 4]
  ```

3.2.1. Tipo Subrango Python

- **range(m, n) = [m, m+1, ... , n-1]**

>>> range(5, 10) [5, 6, 7, 8, 9] >>> list(range(-5, 1)) [-5, -4, -3, -2, -1, 0]

Si n es menor o igual que m, se crea una lista vacía.

>>> list(range(5, 1)) [] >>> list(range(3, 3)) []

- **range(m, n, p) y crea una lista que empieza en m y acaba antes de llegar a n, avanzando de p en p**

>>> list(range(5, 21, 3)) [5, 8, 11, 14, 17, 20] >>> list(range(10, 0, -2)) [10, 8, 6, 4, 2]

Resumen:

m: el valor inicial

n: el valor final (que no se alcanza nunca)

p: el paso (la cantidad que se avanza cada vez).

Si se escriben sólo dos argumentos, Python le asigna a p el valor 1.

range(m, n) es lo mismo que range(m, n, 1)

Si se escribe sólo un argumento, Python, le asigna a m el valor 0 y a p el valor 1.

range(n) es lo mismo que range(0, n, 1)

3.2.2. Ejemplo Tipo Subrango Python

Ejemplo

- **Mostrar un mensaje indicando el tipo de carácter introducido por teclado.**

```
def subrango(car):
    '''car<- retorna mensaje'''
        car = input('introduzca un caracter: ')
        n=ord(car)
        if n in range(ord('A'),ord('Z')+1):
            print('El carácter introducido es una letra mayúscula')
        elif n in range(ord('a'),ord('z')+1):
            print('El carácter introducido es una letra minúscula')
        elif n in range(ord('0'),ord('9')+1):
            print('El carácter introducido es un número')
        else:
            print('El carácter introducido es un carácter especial') #Modificar código para Ñ ó ñ
```

3.3. Tipo vector

- Un vector es una agrupación ordenada, limitada y homogénea de datos del mismo tipo. En memoria se almacenan los datos en posiciones consecutivas.

  Por ejemplo: una lista de enteros podría tener los valores:

  2, 3 ,4 ,5; que es distinto del vector 5, 4 ,3 ,2

- Sintaxis:

  identVector = vector [*subrango*] de *tipo*

- En un vector intervienen un identificador (nombre), una dimensión (número de datos que almacena o agrupa, generalmente expresada mediante un subrango), un tipo de datos y una serie de datos de dicho tipo.

3.3. Tipo Vector

Tipo Vector

- Ejemplos: const max=5;

```
Tipo        tvectorEnteros: vector [1..max] de entero
            tVectorLetras : vector [1..max] de caracter
            Tipo_Vector : vector ['A'..'E'] de real
Variable        mivector: tvectorEnteros
                mivector2: tvectorLetras
                mivector3: Tipo_Vector
```

- Para poder consultar el valor de un determinado dato del vector se debe conocer la posición que ocupa dentro del mismo y el nombre de dicho vector para poder referenciarlo.
- El valor de una determinada posición se obtiene utilizando la siguiente sintaxis:

Identificador_*variable_vector* [*posición*].

3.3. Tipo Vector

Posiciones	0	1	2	3	4
	miVector [0]	miVector [1]	miVector [2]	miVector [3]	miVector [4]
Valores de miVector	**1**	**8**	**2**	**5**	**7**
	miVector2 [1]	miVector2 [2]	miVector2 [3]	miVector2 [4]	miVector2 [5]
Valores de miVector2	**‘D’**	**‘g’**	**‘z’**	**‘*’**	**‘€’**

Posiciones	‘A’	‘B’	‘C’	‘D’	‘E’
	miVector3 [‘A’]	miVector3 [‘B’]	miVector3 [‘C’]	MiVector3 [‘D’]	MiVector3 [‘E’]
Valores de miVector3	**2.5**	**8.9**	**-45.2**	**3.0**	**-100.56**

3.3. Tipo Vector

Manejo de vectores

Sean las definiciones

```
Constante    Max = 10
Tipo         tvectorEnteros: vector [1..max] de entero
Variable     mivector: tvectorEnteros
```

Realizaremos la siguientes operaciones para el manejo de vectores:

- Asignación de un dato a una posición concreta.

  mivector[3] ← 65

- Lectura y escritura de los elementos de un vector.
- Búsqueda, recorrido o acceso secuencial, consiste en pasar por todas las posiciones del vector para procesar su información (veremos ej. En Python).

3.3. Tipo Vector

Lectura *de los datos de un vector. Dimensión completa*

```
desde i<-min hasta max hacer
    leer (mivector[i])
fin_desde
```

Escritura de los datos del vector. Dimensión completa

```
desde i<-min hasta max hacer
    escribir (mi_vector[i], ' ; ' ,)
fin_desde
```

Si el vector no esta completo, nº total de datos viene dado por fin_datos

La posición final será fin_datos en lugar de max.

3.3. Tipo Vector

Tipo Vector

Posiciones	1	2	3	4	5
	miVector [1]	miVector [2]	miVector [3]	miVector [4]	miVector [5]
Valores de miVector	**1**	**8**	**2**	**5**	**7**
	miVector2 [1]	miVector2 [2]	miVector2 [3]	miVector2 [4]	miVector2 [5]
Valores de miVector2	**'D'**	**'g'**	**'z'**		

Fin datos<-3

Fin datos<-'D'

Posiciones	'A'	'B'	'C'	'D'	'E'
	miVector3 ['A']	miVector3 ['B']	miVector3 ['C']	MiVector3 ['D']	MiVector3 ['E']
Valores de miVector3	**2.5**	**8.9**	**-45.2**	**3.0**	

3.3.1. Tipo Vector Python

- En Python se pueden simular el tipo array que recordemos sirve para almacenar elementos del mismo tipo. Para ello, previamente hay que instalar y llamar un módulo llamado numpy.
- Por tanto para trabajar con arrays en Python necesitaremos:

1º- Primero instalamos librería NUMPY:

http://www.cdlibre.org/consultar/catalogo/Python_Bibliotecas.html Desde aquí se accede a NumPy 1.9.2 (py 3.4), nos re direcciona a otra página para su ejecución.

2º- Una vez instalada la librería, si necesitamos en un programa trabajar con arrays necesitaremos obligatoriamente escribir dos instrucciones:

#1. Invocar a la librería

import numpy **as** np

NOTA1: No confundir arrays con listas.

NOTA2.: Los programas no funcionarán en Python tutor, ya que tendríamos que incorporar la librería NUMPY

3.3.1. Tipo Vector Python

#2. En el cuerpo principal inicializando el vector:

v = np.empty((Max,), dtype=np.int)

#Mediante esta función inicializo todas las componentes del vector CON BASURA

#Max es una constante que especifica la dimensión máxima del vector

#dtype me detalla el tipo de datos del vector

- **Los vectores pueden tener diferentes tipos de elementos, esto se especifica mediante atributo dtype**

dtype=np.int #especificación del tipo para crear vectores de enteros

dtype=np.float # especificación del tipo para crear vectores números reales

dtype=np.bool # especificación del tipo para crear vectores de tipo booleano

dtype=np.str # especificación del tipo para crear vectores de tipo carácter

dtype=np.object# especificación del tipo para crear vectores de tipo cadena de caracteres

Nota: Tambien se podría incializar a ceros pero daría errores si el tipo de datos no es numérico y si alguna componente del vector es cero

v = np.zeros((Max,), dtype=np.int) #Mediante esta función inicializo a ceros solo vale para vectores numéricos

2. Los vectores de registros se especificara como se crean en el apartado 3.4.

3.3.1.1. Lectura y escritura vector Python

```
import numpy as np
Max = 10 #Constante para indicar dimensión del vector
def leer(v,n):
    for i in range(n):
        v[i] = int(input('Introduzca el número'))
     return v

def escribir(v,n):
    for i in range(n):
        print(v[i])

v = np.empty((Max,), dtype=np.int) #Inicializa un vector de enteros hasta la Posición Max, si quieres otro tipo se modifica dtype
n = int(input('Dime la dimensión del vector')) #La dimensión debe ser mayor que 0 y menor que la constante(CREAR SUBRUTINA)
v= leer(v,n)
escribir(v,n)
```

3.3.1.2. Lectura datos vector preguntando usuario

Lectura de los datos del vector, preguntando al usuario.

```
import numpy as np
#definición constantes
Min = 0
Max = 15
#Definición subrutinas
def seguir():
    while True:
        respuesta = input('¿Desea continuar( S/N)?')
        if respuesta.upper() == 'S' or respuesta.upper() == 'N':
            break
    return respuesta.upper() == 'S'
```

3.3.1.2. Lectura datos vector preguntando usuario

```
def leer_Vector (mivector):
    ''' array <- array, dimension  Introduce un vector vacío y lo rellena con las componentes que se
        deseen'''
    i = Min
    fin_datos = 0
    b = True
    while b and i <= Max:
        mivector[i] = int(input('Dame un nuevo elemento: '))
        i = i+1
        fin_datos = fin_datos+1
        if  i <= Max:
            b = seguir()
        else:
            print('el vector ya está completo')
    return mivector, fin_datos
```

3.3.1.2. Lectura datos vector preguntando usuario

```
def escribir(v,n):
    for i in range(n):
        print(v[i])

#Cuerpo del programa
v = np.empty((Max,), dtype=np.int) #Inicializa un vector de enteros hasta la Posicion Max
v,n = leer_Vector(v)
escribir(v,n)
```

3.3.1.3 Búsqueda del Máximo y mínimo elemento de un vector

```
def buscar_mayor_y_menor(v,n):
    maximo = v[0]; # Repetir ejercicio importando libreria sys,  mediante sys.maxsize
    minimo = v[0];
    for i in range(1,n):
        if v[i] > maximo:
            maximo = v[i]
        else:
            if v[i] < minimo :
                minimo = v[i]
    return maximo, minimo
#Cuerpo del programa
v = np.empty((Max,), dtype=np.int) #Inicializa un vector de enteros con dimensión  Max
n = int(input('cuantos elementos quieres que tenga el vector: ))# Repetir  teniendo en cuenta dimensiones mediante función
v = leer(v,n)
escribir(v,n)
m,n = buscar_mayor_y_menor(v,n)
print(' El mayor es ',m,' el menor es ',n)
```

3.3.1.4 Otras operaciones con vectores Python

Recorrer al revés un vector

Inserción de datos: consiste en introducir un elemento en el interior de un vector.

Añadir datos: es un caso especial de la operación de inserción de un elemento en un vector, pero el elemento lo introducimos después de la última posición que contiene información válida [fin_datos].

En ambos casos se actualiza el tamaño del vector: **fin_datos← fin_datos+1**

Borrar datos:

- Para eliminar un elemento de un vector si ese elemento está posicionado al final del vector, simplemente se actualiza el tamaño del vector fin_datos← fin_datos-1
- Si el elemento a borrar ocupa cualquier otra posición entonces se desplazan todos los elementos situados a la derecha del que quiero borrar una posición hacia la izquierda y se

actualiza el tamaño del vector: **fin_datos← fin_datos-1**

3.3.1.4.1. Recorrer elementos del vector orden inverso Python

Recorrer una vector al revés en Python

```
import numpy as np

letras = ['m', 'o','c']
vector = np.array(letras)
for i in range(len(vector)-1, -1, -1):
    print(vector[i], end=" ")
```

3.3.1.4.2. Insertar un elemento en una posición de un vector

```
import numpy as np
Max = 10 #Constante del programa para
inicializar

def leer(v,n):
    for i in range(n):
        v[i] = int(input('Introduzca el número: '))
    return v

def escribir(v,n):
    for i in range(n):
        print(v[i])
```

```
def insertar_dato ( v, posicion,dato,f):
    if f < Max:
        if posicion < f + 1:
            p = posicion
            for i in range(f+1,posicion-1,-1):
                v[i+1] = v[i]
            v[p] = dato
            f = f+1
            return v,f
    else:
        print(' no se puede insertar datos el vector esta completo')

v = np.empty((Max,), dtype=np.int) #Inicializa vector dim.  Max
n = int(input('cuantos elementos quieres que tenga el vector: '))
v = leer(v,n)
escribir(v,n)
p = int(input('dime la posición donde insertar el dato: '))
dato = int(input('Dame el dato a insertar: '))
v,f = insertar_dato(v,p,dato,n)
escribir(v,f)
```

3.3.1.4.3.Eliminar un elemento en una posición de un vector

```
import numpy as np
Max = 10 #Constante del programa para inicializar
def leer(v,n):
    for i in range(n):
        v[i] = int(input('Introduzca el número: '))
    return v

def escribir(v,n):
    for i in range(n):
        print(v[i])
```

```
def eliminar_dato (v, posicion, f):
    #Suponemos que el vector esta creado hasta la posición fin_datos(f)

    if posicion < f:
        for i in range(posicion,f-1):
            v[i] = v[i+1]
    v[f] = 0
    f = f-1
    return v,f
v = np.empty((Max,), dtype=np.int) #Inicializa vector hasta Pos. Max
n = int(input('cuantos elementos quieres que tenga el vector: '))
v = leer(v,n)
escribir(v,n)
p = int(input('dime la posición donde eliminar el dato: '))
v,f = eliminar_dato(v,p,n)
escribir(v,f)
```

3.3.2. Vectores multidimensionales. Matrices

- Los vectores ***Multidimensionales*** se utilizarán cuando se necesite manejar grupos de datos homogéneos agrupados en forma de tabla o matriz con dos o más subíndices (matrices multidimensionales).
- Los más utilizados son las matrices (vectores de dos dimensiones). Éstos se pueden considerar como vectores de vectores (vectores donde el tipo de datos de cada elemento es otro vector). Siempre se manejan con dos subíndices.
- Sintaxis :

TIPO

identMatriz = vector [*subrango1*, *subrango2*, ..., *subrangoN*] de *tipo*

3.3.2. Vectores multidimensionales. Matrices

- Ejemplo

	1	2	3	4	5	6	7	8	9	10
1	1	2								
2		4						16		
...	...	...	...	...	...	...	...	...	...	...
10				40					90	

- A los valores de los vectores ***Multidimensionales*** se accede directamente, como en los vectores unidimensionales, solo que utilizando dos o mas subíndices, tantos como dimensiones.
- En el caso de matrices sería M [i, j], donde el primer subíndice hace referencia a las filas y el segundo a las columnas.

3.3.2. Vectores multidimensionales. Matrices

Declarar un tipo de datos válido para representar la temperatura media de los días de un año

Tipo

meses = (enero, febrero, marzo, abril, mayo, …., noviembre, diciembre)

temperaturas = vector [enero..diciembre, 1..30] de real

Variable

medias: temperaturas

medias [octubre, 25] ← 18,3

Se podría añadir una tercera dimensión que representara los años:

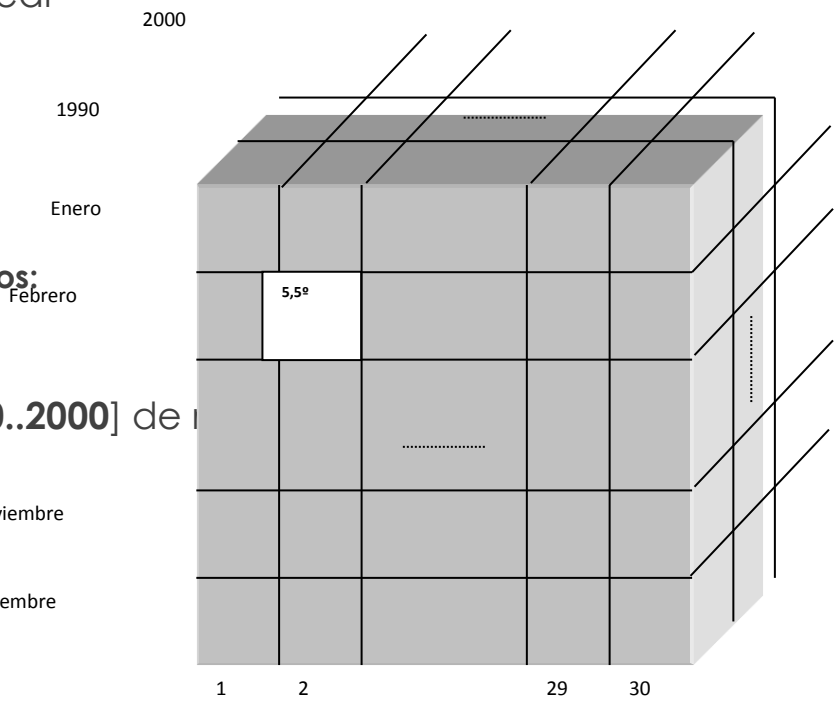
Tipo

temperaturas = vector [enero..diciembre, 1..30, **1990..2000**] de r

Variable

medias: temperaturas

medias [febrero, 2, 1990]← 5,5



3.3.2. Vectores multidimensionales. Matrices

Manejo de matrices

Sean las definiciones

```
Constantes
    N1=   ;
    N2=   ; {se puede dar cualquier valor a las constantes}
Tipo
tdimension1=1..N1;
tdimension2= 1..N2;
     Matriz=vector[1..N1,1..N2] de enteros ;
Variable
    M: Matriz;
    i,j:entero;
```

Realizaremos la siguientes operaciones para el manejo de matrices:

- Asignación de un dato a una posición concreta.
  M[2,8] ← 16
- Lectura y escritura de los elementos de una matriz.
- Búsqueda, recorrido o acceso secuencial

3.3.3. Matrices Python

- En Python se pueden simular el tipo matriz. Para ello, previamente hay que instalar y llamar a numpy.
- Por tanto para trabajar con arrays multidimensionales en Python necesitaremos:

  1º- Tener instalada la librería NUMPY:

  2º- Una vez instalada la librería, si necesitamos en un programa trabajar con matrices se tendrá que escribir estas instrucciones:

  #1. Invocar a la librería

  ```
  import numpy as np
  ```

  #2. En el cuerpo principal inicializar la matriz

  ```
  m = np.empty((Max,Max,), dtype=np.int)
  ```

  #Mediante esta función inicializo todas las componentes de la matriz CON BASURA

  #*(Max,Max,)* constantes que especifican la dimensión máxima de las filas y columnas de la matriz vector

  #dtype me detalla el tipo de datos de la matriz (Igual que en arrays unidimensionales)

3.3.3.1. Lectura de matrices Python

```
#Bloque declarativo
import numpy as np
Max = 10 #Constante del programa para inicializar


def leer(m,nf,nc):
    for i in range(nf):
        for j in range(nc):
            m[i][j] = int(input('Introduzca el número: '))
    return m
```

3.3.3.2. Escritura de matrices Python

```
#Visualización de los datos de una matriz
def escribir(m,nf,nc):
    for i in range(nf):
        for j in range(nc):
            print(m[i][j],end=' ')
        print()

#Cuerpo Principal del programa
m = np.empty((Max,Max,), dtype=np.int) #Inicializa matriz de dimensión fila y columna Max
f = int(input(' Dime el número de filas'))
c = int(input(' Dime el número de columnas'))
m = leer(m,f,c)
escribir(m,f,c)
```

3.3.3.3. Suma de matrices

```
def sumar_matrices (m1,m2,msuma,nf,nc):
    for i in range(nf):
        for j in range(nc):
            msuma[i][j] = m1[i][j] + m2[i][j]
    return msuma
#Cuerpo principal
    m1 = np.empty((Max,Max,), dtype=np.int) #Inicializa matriz de enteros de dimensión  Max filas y columnas  con basura
    m2 = np.empty((Max,Max,), dtype=np.int)
    s = np.empty((Max,Max,), dtype=np.int)
    f = int(input(' Dime el número de filas')) # Realizar  función para solicitar número de filas y columnas
    c = int(input(' Dime el número de columnas'))
    m1 = leer(m1,f,c)
    print('Segunda matriz')
    m2 = leer(m2,f,c)
    suma = sumar_matrices(m1,m2,s,f,c)
    escribir(suma,f,c)
```

3.3.3.3. Búsqueda de un elemento en Matriz

```
def buscar_dato  (m,nf,nc, valor):
    '''Busca en un a matriz (nfxnc) un valor introducido por el usuario retorna las posiciones  donde aparece '''

    cuantas =  0
    for i in range(nf):
        for j in range(nc):
            if m[i][j] == valor:
                cuantas = cuantas + 1
                print('El valor introducido esta en la posición ',i,'  ',j)
    print('El valor aparece', cuantas, 'veces')


#Cuerpo del programa
valo = int(input('Dime valor que buscas'))
buscar_dato(m,f,c,valo)
```

3.3.3.4. Búsqueda booleana de un elemento en matriz

```
def buscar_un_dato ( m,nf,nc,valor):
    '''Dada una matriz m[nf,nc] y un valor indica si el valor está en la matriz'''
    encontrado = False
    i = 0
    while (i<=nf) and (not encontrado):
        j = 0
        while (j<=nc) and (not encontrado):
            if (m[i][j] != valor):
                j = j +1
            else:
                encontrado = True
        if (not encontrado):
            i = i +1
    return encontrado
```

3.3.3.5.Búsqueda posicional de un elemento en matriz

```
def buscar_posicion_dato (m,nf,nc,valor):
'''Dada matriz m[nf,nc] y un valor retorna primera posición donde se encuentra un elemento'''
  encontrado = False
  i = 0     j = 0
  while (i<=nf) and (not encontrado):
      while (j<=nc) and (not encontrado):
          if (m[i][j] != valor):
              j = j +1
          else:       encontrado = True
      if (not encontrado):
          i = i +1
 if  encontrado:
     return i,j
  else:
     return -1,-1 # Posición ficticia indicaría que el  elemento no está en la matriz
```

TRABAJO PERSONAL

Trabajo personal

Lectura recomendada

http://labdls.blogspot.com.es/2014/11/instalar-librerias-modulos-de-python-en.html

https://computacion11fisica.files.wordpress.com/2009/10/arreglos-matrices.pdf

http://progra.usm.cl/apunte/materia/arreglos.html

http://www.maestrosdelweb.com/guia-python-secuencias/ #**IMPORTANTE LISTAS Y TUPLAS(otros tipos de datos)**

Laboratorio: Hoja Tema 3 parte 1.

Resolver ejemplos con Python :

1. Suma de 2 matrices.
2. Dada una matriz y un elemento ver si existe el elemento y la posición que ocupa en la matriz.

3.4 Cadenas de caracteres

- Sucesiones de caracteres cuya longitud máxima, en la mayoría de los lenguajes estructurados, es de 255 caracteres.
- Al declarar un variable de este tipo es conveniente limitar el tamaño de la estructura de datos para evitar un excesivo consumo de memoria.
- Sintaxis en lenguajes estructurados:

tipo *identTipo* = cadena_de_caracteres[MAX]

{ MAX es una constante que indica la dimensión máxima de la cadena}

También se puede declarar la variable directamente (no recomendado)

var identVble: cadena_de_caracteres[MAX]

- Ejemplos:

```
tipo
    apellidos= cadena_de_caracteres[30];
    nombre=cadena_de_caracteres[10];
    telefono= cadena_de_caracteres[10];
```

3.4 Cadenas de caracteres

- En los lenguajes de alto nivel es posible utilizar directamente las sentencias de asignación, lectura y escritura.
- En general, en los lenguajes estructurados, es necesario tener en cuenta la longitud declarada para la cadena.
  - Ejemplo:

    Tipo **cadena = cadena_de _caracteres[10]**

    Variable **cad = cadena**

    Inicio

    cad <- 'Esta es mi primera cadena'

    escribir (cad)

    Fin

    Salida por pantalla será:

    'Esta es mi'

    Debido a que la variable cad es de tipo cadena y su longitud máxima es de 10 caracteres con lo cual no se habrá almacenado nada más que los 10 primeros caracteres de 'Esta es mi primera cadena' en memoria

3.4.1 Cadenas de caracteres (Python)

- En Python una cadena es una secuencia de caracteres (letras, números, espacios, marcas de puntuación, etc.) se distingue porque va *encerrada entre comillas simples o dobles*.
- En Python al no declarar los tipos de datos previamente no podemos fijar de antemano la dimensión de una variable de este tipo (al contrario que en los lenguajes estructurados)
- Ejemplos:

```
miCadena = input('Dame un texto de cinco caracteres: ')
print('El texto leido es: ', miCadena,' de longitud: ', len(miCadena))
```

El ejemplo anterior dada las siguientes entradas, obtendrá las salidas:

Entrada: **'reloj'** → Salida: 'El texto leído es: reloj de **longitud: 5'.**

Entrada: '**casa'** → Salida: 'El texto leído es: casa de **longitud: 4'.**

Entrada: **'relojes**' → Salida: 'El texto leído es: reloj de **longitud: 7'.**

3.4.1 Cadenas de caracteres (Python)

- Una vez que una variable de este tipo **tiene valor**, es posible hacer referencia a cada una de sus posiciones individualmente.
- Para ello se utilizará el identificador de la variable y el número de la posición, entre uno y la longitud de la cadena, entre corchetes.

```
>>> Obtener los caracteres que ocupan la posición 0,3 y 4
    miCadena = 'Esta es la cadena'
    print(miCadena[0],' ', miCadena[3],'  ', miCadena[4])
#Salida: 'E' 'a' ' '
>>> Sustituir en la cadena anterior el carácter 'a' por 'o'
    for i in range(len(miCadena)-1):
        if miCadena[i] == 'a':
                miCadena = miCadena[:i]+'o'+miCadena[i+1:]
    print(miCadena)
#Salida: 'Esto es mi codeno'
```

3.4.1 Cadenas de caracteres (Python)

2ª Forma mediante función:

>>> Sustituir en la cadena anterior el carácter ‘a’ por ‘o’ utillizando el método replace

```
miCadena = 'Esta es la cadena'
print(miCadena[0],' ', miCadena[3],'  ', miCadena[4])
print(miCadena.replace('a', 'o'))# sin recorrer no simula lenguajes estructurados
#Salida: ‘Esto es mi codeno’
```

3.4.2 operaciones con cadenas de caracteres Python

- **Concatenar o sumar**

```
>>> "Un divertido "+“lenguaje "+"de "+ “Programación"
    'Un divertido lenguaje de  Programación‘
```

- **Multiplicar:**

```
>>> 2 * "programas “
    'programas programas '
>>> "programas " * 3
    'programas programas programas '
```

3.4.2 operaciones con cadenas de caracteres Python

- **Longitud** de una cadena de caracteres.el resultado es un nº :

```
>>> len("programas ")
        10
```

- **Segmentación de cadenas:**

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

>>>word ='Python'

>>>word[0:2] # Carácteres desde posición 0 (incluida) hasta 2 (excluida) 'Py'

>>> word[2:5] # Carácteres desde posición 2 (incluida) hasta 5 (excluida) 'tho'

>>> word[:2] # Carácteres desde posición 0 hasta la 2 (excluida) 'Py'

>>> word[4:] # Carácteres desde posición 4 (incluida) hasta el final 'on'

>>> word[-2:] # Carácteres desde la penúltima posición (incluida) to the end 'on'

3.4.2 operaciones con cadenas de caracteres Python

- **in** Para buscar una subcadena (o un caracter) en una cadena, Python nos ofrece varias alternativas. Si solamente necesitamos saber si una cadena contiene cierto caracter o cierta subcadena, usamos el operador in

  >>> t in aeiou (False)

  >>> Tutorial in videotutorial (True)

- **ord(**cadena**)** Dada una cadena retorna su código Unicode(un entero)

  >>>ord('a')# retorna 97

- **chr(**entero**)** Dado una entero retorna una cadena que tiene a ese entero como código Unicode

  >>>chr(97) #retorna 'a'

3.4.2 operaciones con cadenas de caracteres Python

- **a.lower ()**(paso a minúsculas): retorna una cadena con los caracteres de a convertidos en minúsculas .

  >>> a =‘AsA’

  >>>a.lower() #retorna ‘asa’

- **a upper ()**(paso a mayúsculas): retorna una cadena con los caracteres de a convertidos en mayúsculas.

  >>> a =‘AsA’

  >>>a.upper() #retorna ‘ASA’

- **a.capitalize()** sirve para convertir el primer carácter de un string a mayúsculas

  >>> a =‘hola mundo’

  >>>a.capitalize() #retorna ‘Hola mundo’

3.4.2 operaciones con cadenas de caracteres Python

- **a title ()**: retorna una cadena en la que toda palabra de a empieza por mayúscula.

  >>> a =‘hola mundo’

  >>>a.title() #retorna ‘Hola Mundo’

- **a.replace(‘cad1,cad2**) Reemplaza en la cadena a la subcadena cad1 por cad2. replace() no altera el valor de la variable sobre la que se ejecuta

  >>> a= "Esto será reemplazado: hola”

  >>> print (a.replace('hola', 'mundo'))

  SALIDA: Esto será reemplazado: mundo **# NOTA: El valor de variable a sigue siendo el mismo “Esto será reemplazado:hola**

- **strip(), lstrip() y rstrip()** nos ayudan a eliminar todos los espacios en blanco, solo los que aparecen a la izquierda y solo los que se encuentran a la derecha, respectivamente

  >>> cadena = " esta cadena tiene espacios a los lados “

  >>> cadena.strip() # retorna : "esta cadena tiene espacios a los lados“

  >>> cadena.lstrip() # retorna “ esta cadena tiene espacios a los lados “

  >>> cadena.rstrip() # retorna " esta cadena tiene espacios a los lados"

3.4.2 operaciones con cadenas de caracteres Python

- **format(*args, **kwargs)** Da formato a una cadena, sustituyendo texto dinámicamente, retorna la cadena formateada

```
>>> cadena = "bienvenido a mi aplicación {0}"
>>>print (cadena.format("en Python") ) # retorna: bienvenido a mi aplicación en Python
>>> cadena = "Importe bruto: ${bruto} + IVA: %{iva} = Importe neto: {neto}"
>>>print( cadena.format(bruto=100, iva=21, neto=121))
                # retorna: Importe bruto: $100 + IVA: %21 = Importe neto: 121
>>>print(cadena.format(bruto=100, iva=100 * 21 / 100, neto=100 * 21 / 100 + 100))
                # retorna: Importe bruto: $100 + IVA: $21.0 = Importe neto: 121.0
```

- **a.find()** regresa el índice correspondiente al primer carácter de la cadena original que coincide con lo que estamos buscando:

```
>>>a = "Esto será reemplazado: hola"
>>>print(a.find('r')) # retorna 7
>>>print(a.find('x'))#retorna -1
```

- **Operadores relacionales** para comparación de cadenas de caracteres. == , !=, < ,>

3.4.3 Ejemplos con cadenas de caracteres Python

1.Dada una cadena de caracteres, reemplazar los dígitos existentes en la misma por un guion.

```
def reemplazar_digitos (cad):
    '''cad<- cad Dada una cadena reemplaza los dígitos existentes por _'''
    subcadena =''
    for i in range(len(cad)):
        if (cad[i] >='0') and  (cad[i] <='9'):
            subcadena = subcadena + '_'
        else:
            subcadena =subcadena + cad[i]
    return subcadena


cad = input('Cadena?')
print(' la cadena: ',cad,' queda sin dígitos: ',reemplazar_digitos(cad))
```

Ejemplo **cad = s34 f5 # retorna: s__ f_**

3.4.3 Ejemplos con cadenas de caracteres Python

2. Insertar una cadena dentro de otra, comenzando en la posición indicada, suponiendo que no disponemos de esta operación.

```
def insertar_subcadena(cadena,cadinsert,posicion):
    cadena2 = cadena[:posicion]+cadinsert+cadena[posicion:]
    return cadena2


#Cuerpo principal
c = input('Cadena inicial: ')
ci= input('cadena a insertar: ')
p = int(input(' Posición en la que insertar: '))
cadena = insertar_subcadena(c,ci,p)
print(' El resultado es:', cadena)
```

```
>>> #Ejecución:
Cadena inicial: HOLA MI ES MARIAN
cadena a insertar: nombre
 Posición en la que insertar: 8
 El resultado es: HOLA MI nombre ES MARIAN
>>>
```

3.4.4 Programa para enviar textos por correo con Python

CURIOSIDADES DE PYTHON

INFOMACIÓN PARA RESOLVER PROBLEMA: https://librosweb.es/libro/python/capitulo_14/envio_de_e_mail_desde_python.html

```
import smtplib
#Importamos el módulo smtplib
#definimos las variables necesarias para el envío del mensaje
#(remitente, destinatario, asunto y mensaje -en formato HTML
remitente = "Desde marian <marian.fernandez@uah.es>"
destinatario = "Marian <atischa@hotmail.com>"
asunto = "E-mal HTML enviado desde Python"
mensaje = """Hola!<br/> <br/>
Este es un <b>e-mail</b> enviando desde <b>Python</b>
"""
 #generamos el e-mail con todos los datos definidos anteriormente
email = """From: %s
To: %s
MIME-Version: 1.0
Content-type: text/html
Subject: %s
%s
"""% (remitente, destinatario, asunto, mensaje)
try:
    #creamos un objeto smtp y realizamos el envío
    smtp = smtplib.SMTP('localhost')
    smtp.sendmail(remitente, destinatario, email)
    print( "Correo enviado" )
except:
    print ("""Error: el mensaje no pudo enviarse.
    Compruebe que sendmail se encuentra
instalado en su sistema""")
```

TRABAJO PERSONAL

Trabajo personal

Lectura recomendada

Cap 5 Apartado 1 Cadenas Libro Introducción Programación Python (Andrés Marzal).

Laboratorio: Hoja Tema 3 parte 2.

Resolver ejemplos con Python empleando la sentencias más adecuada:

1. Concatenar dos cadenas.
2. Insertar una cadena dentro de otra a partir de una posición.

3.5 Registros

Un **registro** es un tipo de dato estructurado que permite agrupar un número finito de datos, denominados *campos*.

A diferencia de los vectores cada dato o *campo* puede ser de un tipo de dato diferente y el acceso a cada uno de ellos se realiza mediante su nombre o identificador y no mediante un índice.

```
Tipo        Identificador_TIPO_registro = Registro
                Id_campo1 : tipo
                Id_campo2 : tipo ....
                Id_campoN : tipo
            Fin
```

Cada campo de un registro se define mediante un nombre o identificador del campo y un tipo, simple o previamente definido.

Los nombres de los campos pertenecientes a un mismo registro deben ser todos distintos, aunque pueden volver a utilizarse en otras partes del algoritmo (variables, campos de otros registros, etc.).

3.5. Registros

ACCESO A LOS CAMPOS DE UN REGISTRO: La forma de acceder a los datos almacenados en una variable de tipo REGISTRO en lenguajes estructurados, es mediante el **identificador de la variable** y a continuación, separado por un **punto**, el **identificador del campo** correspondiente.

Ejemplo: **tipo_alumno=Registro**

nombre: cadena_de_caracteres[max_nombre]

apellidos:tipo_nombre

notas:tipo_notas

Fin

Var **al = tipo_alumno**

Inicio

al.nombre <- 'Pepe'

al.apellidos<- 'Fdez'

al.notas <- 10

Fin

3.5.1 Registros en Python

Para **simular los registros** que se utilizan en lenguajes estructurados en **Python** tendremos:

1º Instalar módulo record (copiándola en la ruta de la carpeta donde tengamos instalado Python (C:\Python34))

2º Invocar el módulo

>>>from record import record

3º Crear un registro vacío en cuerpo principal

```
class identificador_registro(record): #Inicializo Variable
    identificador_1 =   (depende del tipo string=''….)
    identificador_2 =''
    identificador_n = 0
Identificador_variable = identificador_registro()
```

3.5.1.1 Lectura y escritura de registros en Python

1. Programa para almacenar datos de personas:

```
from record import record

def lee_registro_person(dato):
        dato.nombre = input('Dime el nombre: ')
        dato.dni = input(' Dime el DNI: ')
        dato.edad = int(input('Edad: '))
        return dato
def escribir_registro_person(dato):
        print('EL nombre es ',dato.nombre)
        print('EL DNI es ',dato.dni)
        print('La edad es ',dato.edad)
```

```
#Cuerpo principal
class Persona(record): #Inicializo Variable
    nombre =''
    dni =''
    edad = 0
dato = Persona()
dato = lee_registro_person(dato)
escribir_registro_person(dato)
```

3.5.1.2 Lectura de registros de registros en Python

2. Programa para almacenar los datos de una persona con fechas:

```
from record import record
def fechas(f):
    while True:
        f.dia = int(input('Día de nacimiento: '))
        if (f.dia>=1) and (f.dia <=31):
            break
    while True:
        f.mes = int(input('Mes de nacimiento: '))
        if (f.mes>=1) and (f.mes <=12 ):
            break
    while True:
        f.año = int(input('Año de nacimiento: '))
        if (f.año>=1900) and (f.año <=2100):
            break
    return f.dia,f.mes,f.año
```

```
def lee_registro_person(dato):
    dato.nombre = input('Dime el nombre: ')
    dato.dni = input(' Dime el DNI: ')
    dato.edad = int(input('Edad: '))
    f = Fecha()
    dato.fecha_nac.dia,dato.fecha_nac.mes,dato.fecha_nac.año =fechas(f)
    return dato
```

3.5.1.3 Escritura de registros de registros en Python

```
def escribir_registro_person(dato):
    print('El nombre es ',dato.nombre)
    print('EL DNI es ',dato.dni)
    print('La edad es ',dato.edad)
    print('Dia ',dato.fecha_nac.dia)
    print('Mes ',dato.fecha_nac.mes)
    print('año ',dato.fecha_nac.año)
```

```
#Cuerpo principal
class Fecha(record):#Inicializo registro Fecha
    dia = 1
    mes = 1
    año = 1900
class Persona(record): #Inicializo registro Persona
    nombre =''
    dni =''
    edad = 0
    fecha_nac = Fecha() #Creo variable tipo registro vacío fecha
dato = Persona() #Creo variable tipo registro vacío Persona
n = int(input('Dime la dimensión del vector: '))
lee_registro_person(dato)
escribir_registro_person(dato)
```

3.5.1.4 Vector de registros en Python

3. Programa para almacenar los datos de un conjunto de personas

```
from record import record

import numpy as np

Max = 10 #Constante del programa para inicializar


def lee_registro_person(dato):
    dato.nombre = input('Dime el nombre: ')
    dato.dni = input(' Dime el DNI: ')
    dato.edad = int(input('Edad: '))
    return dato
```

```
def leer(v,n):
    for i in range(n):
        v[i] = Persona()
        lee_registro_person(v[i])
    return v,n
```

3.5.1.4. Vector de registros en Python

```
def escribir_registro_person(dato):
    print('EL nombre es ',dato.nombre)
    print('EL DNI es ',dato.dni)
    print('La edad es ',dato.edad)
def escribir(v,n):
    for i in range(n):
        escribir_registro_person(v[i])
```

```
#Cuerpo principal
class Persona(record):
    nombre =''
    dni =''
    edad = 0
dato = Persona()
v = np.empty([Max,], dtype=Persona) #Inicializa un vector de enteros hasta pos Max
n = int(input('Dime la dimensión del vector: '))
v,n= leer(v,n)
escribir(v,n)
```

3.5.1.5 Vectores de registros con fechas Python

4. Programa para almacenar los datos de un conjunto de personas con su fecha de nacimiento

```
from record import record

import numpy as np

Max = 10 #Constante del programa para inicializar

def fechas(f):

    while True:

        f.dia = int(input('Día de nacimiento: '))

        if (f.dia>=1) and (f.dia <=31):

            break
```

```
while True:
        f.mes = int(input('Mes de nacimiento: '))
        if (f.mes>=1) and (f.mes <=12 ):
            break
    while True:
        f.año = int(input('Año de nacimiento: '))
        if (f.año>=1900) and (f.año <=2100):
            break
    return f.dia,f.mes,f.año
```

3.5.1.5 Vectores de registros con fechas Python

```
def lee_registro_person(dato):
    dato.nombre = input('Dime el nombre: ')
    dato.dni = input(' Dime el DNI: ')
    dato.edad = int(input('Edad: '))
    f = Fecha()
    dato.fecha_nac.dia,dato.fecha_nac.mes,dato.fecha_nac.año =fechas(f)
    return dato
def leer(v,n):
    for i in range(n):
        v[i] = Persona()
        lee_registro_person(v[i])
    return v,n
```

3.5.1.5 Vectores de registros con fechas Python

```
def escribir_registro_person(dato):
    print('El nombre es ',dato.nombre)
    print('EL DNI es ',dato.dni)
    print('La edad es ',dato.edad)
    print('Dia ',dato.fecha_nac.dia)
    print('Mes ',dato.fecha_nac.mes)
    print('año ',dato.fecha_nac.año)
def escribir(v,n):
    for i in range(n):
        escribir_registro_person(v[i])
```

3.5.1.5 Vectores de registros con fechas Python

```
#Cuerpo principal
class Fecha(record):
    dia = 1
    mes = 1
    año = 1900
fecha_nac = Fecha()
dato = Persona()
v = np.empty([Max,], dtype=Persona) #Inicializa un vector de enteros hasta la Posicion Max, si quieres otro tipo se modifica dtype
n = int(input('Dime la dimensión del vector: '))
v,n= leer(v,n)
escribir(v,n)
```

```
class Persona(record):
    nombre =''
    dni =''
    edad = 0
```

3.5.2 Registros variantes

- Registros con estructura flexible, de manera que no siempre contengan el mismo número y tipo de campos, pero dentro de la misma definición de tipo registro. Es decir, la posibilidad de que *diferentes variables del mismo tipo registro no contengan necesariamente los mismos campos.*
- El número y tipo de campos que contiene un **registro variante** depende del valor que tome uno de sus campos, llamado ***campo discriminante***.
- La declaración de un *tipo registro variante* tiene dos partes:
  - **Parte *fija***: Incluye la declaración de todos los campos que son comunes a todas las variables del tipo.
  - **Parte *variante***: Declaración de los campos no comunes y que depende por tanto del valor del *campo discriminante*. Comienza con la declaración del campo discriminante mediante un identificador y el tipo de dato (siempre ordinal) según la siguiente sintaxis:

    Segun Identificador_campo_discriminante: tipo_ordinal **hacer**.
- A continuación se detallan todos los posibles valores que puede tomar dicho campo, y se declara para cada uno de ellos (entre paréntesis) los campos que existen en el caso de que el *campo discriminante* tome dicho valor.

3.5.2 Registros variantes

la sintaxis general de la declaración de un registro variante es la siguiente:

Tipos

Identificador_registro_variante= **Registro**

Id_campo1: tipo
Id_campo2: tipo
..... {Parte fija}

Segun Id_campo_discriminante: tipo_ordinal **Hacer**

valor1: (Id_campo11:tipo, Id_campo12:tipo,...)

.....

valork: (Id_campok1:tipo, Id_campok2:tipo,...)

Fin {Parte variante}

3.5.2 Registros variantes

- La parte variante siempre se declara después de la parte fija, no se puede incluir ningún campo fijo después de la declaración del campo discriminante. De esta forma el fin indica el final de la declaración del registro y de la parte variante de éste.
- No debe confundirse el uso de la sentencia SEGUN que da comienzo a la parte variante de un registro variante con la sentencia selectiva que tiene el mismo nombre.
- El selector de la sentencia SEGUN de un registro variante no es una variable sino la declaración de un **campo** del registro, cuyo tipo es siempre ordinal.
- Es importante asegurarse de que el campo discriminante de un registro variante tenga valor, fijando así el número y tipo de los campos variantes del mismo.

3.5.2 Registros variantes

EJEMPLO

```
Tipos
  Tipo_empleado=REGISTRO
        Nombre: cadena_de _caracteres [max]
        Salario: real
        SEGUN estudios : caracter HACER
          'U':(titulo: cadena_de _caracteres [max], universidad: cadena_de _caracteres [max])
          'B':(centro: cadena_de _caracteres [max])
          'O':(grado: cadena_de _caracteres [max])
  FIN
```

3.5.2.1 Simulación registros variantes en Python

En Python simularemos este tipo de datos:

1º invocando al módulo record.

2º. declarando tantas variables tipo registro como campos variantes tengamos.

Por tanto, para el ejemplo anterior tendremos que escribir 3 variables tipo registro para los empleados:

```
#1º Registro Estudios Universitarios
 class EmpleadoEstUn(record):
    nombre =''
    salario =0
    titulo = ‘’
    universidad =‘’
Emp1 = EmpleadoEstUn()
```

```
#2ºRegistro Estudios Bachillerato
 class EmpleadoEstBa(record):
    nombre =''
    salario =0
    centro =‘’
Emp2 = EmpleadoEstBa()
```

```
#3º Registro Estudios Obligatorios
 class EmpleadoEstOb(record):
    nombre =''
    salario =0
    grado = ‘’
Emp1 = EmpleadoEstOb()
```

3.5.3 Ejercicio

Escribir un algoritmo que lea los siguientes datos de los alumnos de una clase: nombre, dni, nota en informática y nota en matemáticas. A continuación el algoritmo deberá obtener la nota media del grupo en matemáticas y el nombre y dni del alumno con mejor nota en informática.

```
Algoritmo Ejercicio_1_REGISTROS
  CONSTANTE
        max_alumnos=20
        max_nombre=15

  TIPO
        subrango_alumnos=1..max_alumnos
        tipo_notas=Registro
                    matemáticas, informática:real
        Fin
        tipo_nombre=cadena_de_caracteres[max_nombre]
```

```
tipo_alumno=Registro
                    nombre:tipo_nombre
                    apellidos:tipo_nombre
                    dni:entero
                    notas:tipo_notas
Fin

tipo_alumnos=Registro
                    nombres:vector[subrango_alumnos]
                                de tipo_alumno
                    cuantos:subrango_alumnos
Fin
VARIABLE     alumnos:tipo_alumnos
```

3.5.3 Ejercicio

```
procedimiento lee_datos     (S/ ALUM:tipo_alumnos )
VARIABLE     i: subrango_alumnos
Inicio
    Escribir('Cuantos alumnos hay en el grupo? (maximo:', max_alumnos, ')?')
      Leer(alum.cuantos)
      desde i ← 1 hasta alum.cuantos hacer
        con alum.nombres[i] hacer
              Escribir('Datos del alumno:', i)
              Escribir('nombre') Leer(nombre)
              Escribir('apellidos') Leer(apellidos)
              Escribir('dni') Leer(dni)
            con notas hacer
               Escribir('nota en informática?') Leer(informática)
               Escribir('nota en matemáticas') Leer(matemáticas)
             fin_con
        fin_con
      fin_desde
Fin
```

3.5.3 Ejercicio

```
Funcion media_matematicas (E/ alum:tipo_alumnos):real
variable        suma:real
                i: subrango_alumnos
Inicio
    suma ← 0
    desde i ← 1 hasta alum.cuantos hacer
            con alum.nombres[i], notas hacer
                    suma ← suma+ matemáticas
            fin_con
    fin_desde
            media_matematicas ← suma/alum.cuantos
Fin
```

3.6.4 Registros variantes

```
Función mejor_informática (E/ alum:tipo_alumnos ):tipo_nombre
        var iable    pos_mejor, i : subrango_alumnos
             nota_mejor :real
      Inicio
          pos_mejor ← 1
          nota_mejor← alum.nombres[1].notas.informatica

          desde i ← 2 hasta alum.cuantos hacer
            con alum.nombres[i], notas hacer
                  si informatica> nota_mejor  entonces
                     pos_mejor ← i
                     nota_mejor← informatica
                   fin_si
            fin_con
          fin_desde
          mejor_informática ← alum.nombres[pos_mejor].nombre
      Fin
```

3.5.3 Ejercicio

```
Inicio {Algoritmo principal}
        leer_datos(alumnos)
        Escribir('La nota media de la clase en matemáticas es:', media_matemáticas (alumnos))
        Escribir('El alumno con la mejor nota en informática es:', mejor_informática(alumnos))
Fin
```

TRABAJO PERSONAL

Trabajo personal

Lectura recomendada

Cap 7 Libro Introducción Programación Python (Andrés Marzal).

Laboratorio: Hoja Tema 3 parte 3.

Resolver ejemplos :

1. Pasar el ejercicio del apartado 3.5.3. a Python.