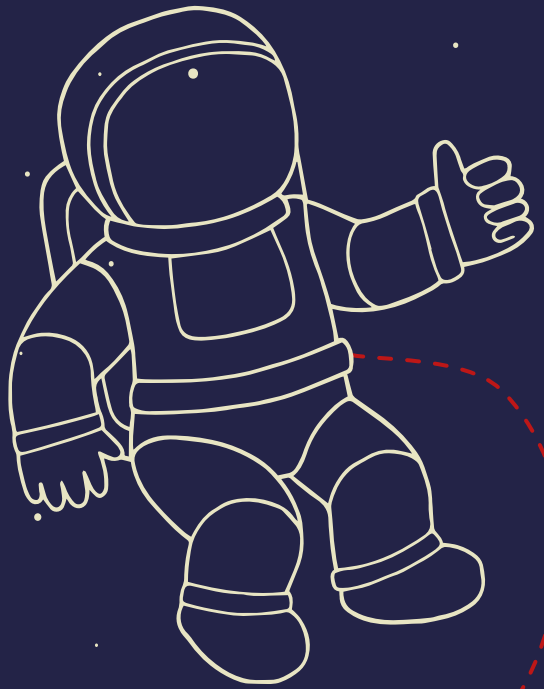
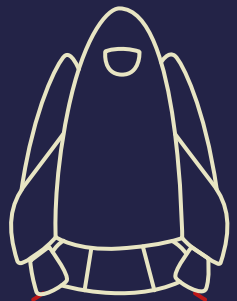
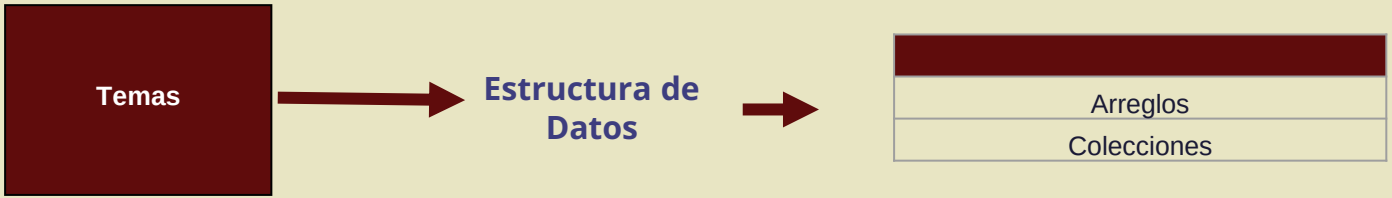


Programa académico CAMPUS

Ciclo 1: Fundamentos de Programación



Presentación Ciclo 1 – Fundamentos de Programación

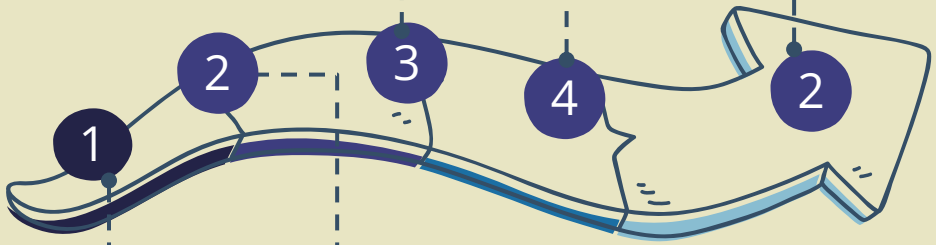


Estructura de Datos

Matrices

Diccionarios

Conjuntos



Listas

Tuplas

Estructura de Datos

Conceptualización

En la vida cotidiana nos vemos enfrentados a crear listas, por ejemplo la lista de útiles para el colegio o la universidad, la lista de personas que se invitará a una fiesta. En la preparación de alimentos, se debe realizar una lista con los ingredientes: carne molida, tomate, pan, cebolla, aceite, queso tajado, lechuga y tocineta.

Las **estructuras de datos** son agrupaciones de variables simples que conforman un conjunto de datos más complejo con el cual puedes dar soluciones eficientes a situaciones más cercanas a la vida práctica, como lo son por ejemplo: el manejo de calificaciones de estudiantes de un curso o la gestión de nómina de los empleados de una empresa.

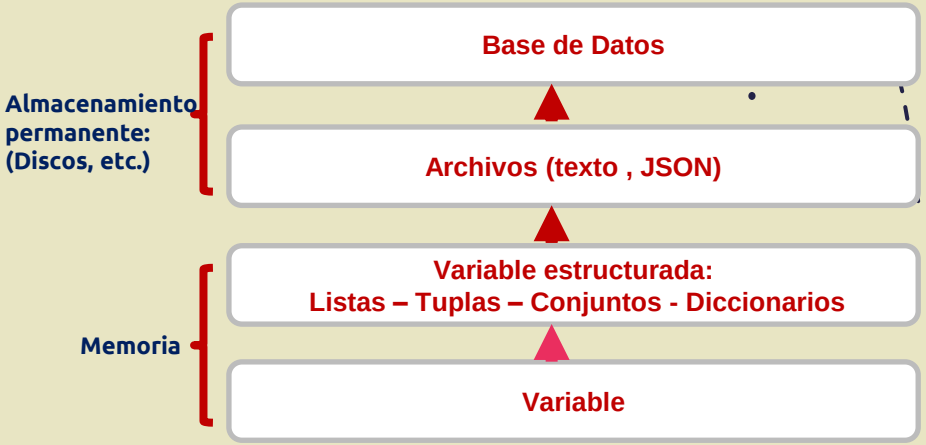
Estructura de Datos

Conceptualización

Variable: Es un espacio de memoria que **contiene un dato simple** de tipo cadena, numérico, booleano, etc.

Al contenido de este espacio de memoria, se accede a través de lo que llamamos un **identificador o Nombre de Variable.**

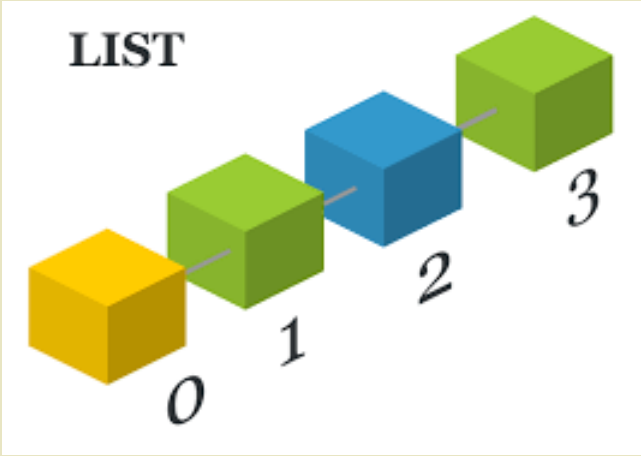
Variable Estructurada: es un agrupamiento, empaquetamiento o colección de varios espacios de memoria, a los cuales se accede a través de un único identificador**.**



Estructura de Datos

Listas

Las listas son estructuras que permiten ser modificadas a lo largo de la ejecución de un programa usando algunos métodos y operadores. Este comportamiento le da la caracterización a las listas de ser **estructuras mutables**.



Estructura de Datos

Listas - Características

Referenciar elementos

nombre_persona[2] = **“Sergio”**

nombre_persona[0:2] = **“Catalina”,”Silvia”**

Estructura de Datos

Listas - Creación

```
>>> lista_numeros=[10,15,20,30,40]
>>> lista_numeros[2]
20
>>> lista_nombres=["Sergio","Catalina","Silvia","Iván","Elsa"]
>>> lista_nombres[4]
'Elsa'
```

```
>>> lista_nombres[0:2]
['Sergio', 'Catalina']
```

```
>>> lista_pares=list(range(2,20,2))
>>> lista_pares
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Crea la lista con valores desde 2 hasta 20, con incrementos de 2. No se toma el valor final del rango

```
>>> lista_elementos=[1,"Juan",[2,3],10.4,"Pedro"]
>>> lista_elementos[2]
[2, 3]
>>> lista_elementos[3:5]
[10.4, 'Pedro']
```

Estructura de Datos

Listas - Métodos

Podemos crear una lista y luego modificar sus elementos mientras se ejecuta el código.

Para esto, las listas tienen un conjunto de métodos y funciones que realizan acciones y operaciones sobre una lista en particular. Algunos de estos métodos son:

append, extend, insert, pop, remove y otros como len para la longitud, es decir número de elementos

El método append permite añadir un ítem al final de una lista

El método extend se utiliza para agregar elementos interables como un string u otra lista separando sus elementos.

El método insert permite añadir un ítem en una posición o índice específico

El método pop quita un elemento de la lista dado su índice.

El método remove para remover un ítem de una lista basado en el valor

Estructura de Datos

Listas - Métodos

```
>>> lista=[10,20,"Juan",30,"Sergio"]
>>> lista
[10, 20, 'Juan', 30, 'Sergio']
>>> lista.append(40)
>>> lista
[10, 20, 'Juan', 30, 'Sergio', 40]
>>> lista.append("Paula")
>>> lista
[10, 20, 'Juan', 30, 'Sergio', 40, 'Paula']
>>> lista.extend([60,80])
>>> lista
[10, 20, 'Juan', 30, 'Sergio', 40, 'Paula', 60, 80]
>>> lista.insert(1,"Luis")
>>> lista
[10, 'Luis', 20, 'Juan', 30, 'Sergio', 40, 'Paula', 60, 80]
>>> lista.pop(4)
30
>>> lista
[10, 'Luis', 20, 'Juan', 'Sergio', 40, 'Paula', 60, 80]
>>> lista.remove("Sergio")
>>> lista
[10, 'Luis', 20, 'Juan', 40, 'Paula', 60, 80]
```

Estructura de Datos

Funciones predefinidas en Python para listas

En los lenguajes de programación, se puede definir cualquier función que se desee; pero, para facilitarnos un poco la vida existen funciones predefinidas, o sea que alguien más ya las creó y fueron incorporadas en el lenguaje de programación, en este caso **Python**.

En **Python**, se pueden encontrar dos tipos de funciones predeterminadas, las cuales son:

- **Funciones predefinidas**.
- **Los módulos predefinidos**. son archivos que contienen **métodos predefinidos** (funciones), que no pueden existir por sí solos, ya que se encuentran asociados a determinado **objeto** o tipo de dato (listas, cadenas, caracteres, etc), de tal manera que, operan sobre ellos.

Estructura de Datos

Funciones predefinidas en Python para listas

```
>>> max(40,30,5,90,20)
90
```

```
>>> min(40,30,5,90,20)
5
```

Funciones

```
>>> numeros=[11,12,31,41,52,63,78]
>>> len(numeros)
7
```

Librerías: Agrupan , funciones, métodos, etc.

```
>>> import math
>>> math.sqrt(16)
4.0
```

Método: Operación

```
>>> numeros=[6,3,1,4,5,2]
>>> numeros_ordenados=sorted(numeros)
>>> numeros_ordenados
[1, 2, 3, 4, 5, 6]
```

```
>>> numeros=[6,3,1,4,5,2]
>>> numeros_ordenados=sorted(numeros,reverse=True)
>>> numeros_ordenados
[6, 5, 4, 3, 2, 1]
```

Estructura de Datos

Funciones predefinidas en Python – Programa

```
# Programa para hallar la raíz cuadrada de un número
# Autor: Sergio Medina
# Fecha: 09/05/2022

import math
x=int(input("Valor: "))
print("Raiz cuadrada de ",x," es: ",math.sqrt(x))
```

Estructura de Datos

Funciones predefinidas en Python para listas

```
>>> numeros=[1,2,3,4,5,6,1,7,1]
>>> print(numeros.count(1))
3
```

```
>>> letras=["a","e","t","v","u","Z","c","a"]
>>> print(letras.count("a"))
2
```

Contar en listas

```
>>> items="1,2,3,4,5,6"
>>> items.split(",")
['1', '2', '3', '4', '5', '6']
>>> items.split(",")[4]
'5'
```

```
>>> items="la casa de Luisa es muy bonita"
>>> items.split()
['la', 'casa', 'de', 'Luisa', 'es', 'muy', 'bonita']
```

Crea Lista desde una cadena , su argumento es el separador de elementos de la lista creada

Cantidad de palabras en una frase

```
>>> items="la casa de Luisa es muy bonita"
>>> items.split()
['la', 'casa', 'de', 'Luisa', 'es', 'muy', 'bonita']
>>> len(items.split())
7
```

Estructura de Datos

Funciones predefinidas en Python – Programa

```
# Programa para calcular la cantidad de palabras de un frase
# Autor: Sergio Medina
# Fecha: 10/05/2022

frase=input("Frase: ")
can_palabras=len(frase.split())
print("Cantidad de palabras: ",can_palabras)
```

Estructura de Datos

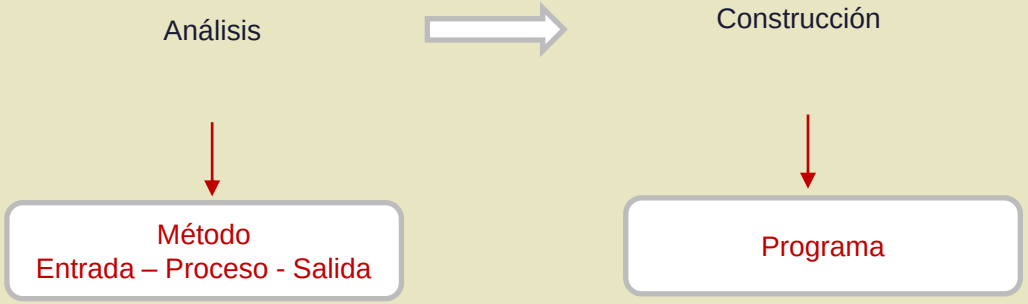
Listas – Ejercicio – Contar Vocales

- Se desea realizar un programa en el cual se ingresen N letras del abecedario, las cuales se deben almacenar en una lista. Una vez creada la lista, se desea conocer e imprimir la cantidad de “a”, la cantidad de “e”, la cantidad de “i”, la cantidad de “o” y la cantidad de “u” que se encuentran en la lista

Estructura de Datos

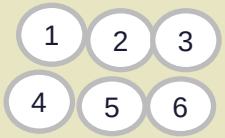
Listas - Ejercicio

Metodología -> Pensamiento lógico estructurado



Estructura de Datos

Listas - Ejercicio



Análisis –> Ejercicio Listas

Lista & Ciclos

- Llenar
- Procesar

Entrada LEER → Proceso → Salida IMPRIMIR

Estructura de Datos

Listas - Ejercicio

```
# Programa para contar vocales de una lista
# Autor: Sergio Medina
# Fecha: 08/06/2022

#iniciar lista como vacias
lista_letras=[]
N=int(input("Cantidad de letras: "))
cantidad_a=0
cantidad_e=0
cantidad_i=0
cantidad_o=0
cantidad_u=0
#LLenar lista
for i in range(N):
    letra=input("Letra: ")
    lista_letras.append(letra)
print("Lista letras: ",lista_letras)
#Procesar lista
for x in lista_letras:
    if x=="a" or x=="A":
        cantidad_a+=1
    elif x=="e" or x=="E":
        cantidad_e+=1
    elif x=="i" or x=="I":
        cantidad_i+=1
    elif x=="o" or x=="O":
        cantidad_o+=1
    elif x=="u" or x=="U":
        cantidad_u+=1
print("Cantidad de a: ",cantidad_a)
print("Cantidad de e: ",cantidad_e)
print("Cantidad de i: ",cantidad_i)
print("Cantidad de o: ",cantidad_o)
print("Cantidad de u: ",cantidad_u)
```

Estructuras de Datos

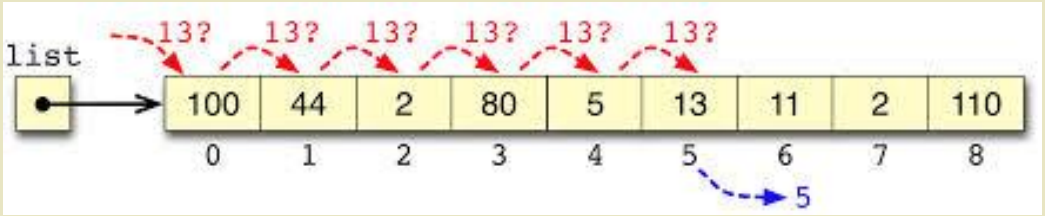
Búsqueda de información

Las **búsquedas de información** consisten en un proceso iterativo en el **que** se realiza una **búsqueda**, se analizan los resultados y se va modificando la estrategia de **búsqueda** hasta identificar los términos y las fuentes de **información que** nos proporcionan los resultados más pertinentes.

Estructuras de Datos

Búsqueda de información

El Método de búsqueda secuencial o lineal, tiene como objetivo buscar una información suministrada dentro de una lista. El proceso de búsqueda secuencial o lineal consiste en comparar el elemento a buscar con cada uno de los elementos de la lista hasta ser encontrado o finalizar la lista.



Estructuras de Datos

Búsqueda de información

Dada una lista vector de N elementos y una información a buscar en la lista, se pide realizar el proceso de búsqueda lineal o secuencial e indicar en cual posición de la lista se encuentra la información.

Entrada:

Lista de números
2
4
1
5
3

Información a buscar: 5

Salida:

Posición donde se encontró la información
3

Estructuras de Datos

Búsqueda de información

Construcción –> Programa

```
        try:
            dato=int(input(etiqueta))
            break
        except ValueError:
            print(etiqueta," debe ser dato ENTERO")
    return dato

def busqueda_lineal(numeros,buscar):
    for i in range(N):
        if numeros[i]==buscar:
            return i
    return -1
```

Estructuras de Datos

Ordenamiento de información

El **ordenamiento** organiza los **datos** en orden alfabético o numérico ascendente o descendente. Por ejemplo, puede ordenar una columna que enumere los valores de las ventas de un producto en orden descendente para ordenar las ventas del producto de la más alta a la más baja.

Estructuras de Datos

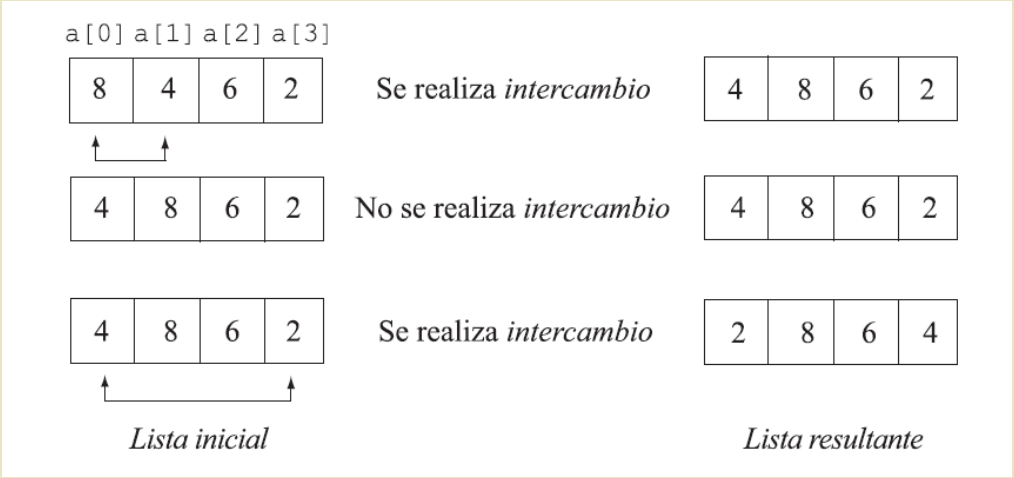
Ordenamiento iterativo – Método Burbuja

El ordenamiento burbuja funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

Estructuras de Datos

Ordenamiento iterativo – Método Burbuja

Ubicar la información de una lista o vector de menor a mayor (Ascendente) o de mayor a menor (Descendente): El método de Burbuja tradicional realiza el ordenamiento a través de dos ciclos anidados y un condicional.



Estructuras de Datos

Ordenamiento iterativo – Método Burbuja

Dada una lista vector de N elementos, se pide ordenarla de menor a mayor (Ascendente) mediante el método de Burbuja tradicional.

Vector de entrada:

Lista de números
5
3
4
2
1

Vector de salida:

Lista de números
1
2
3
4
5

Estructuras de Datos

Ordenamiento iterativo – Método Burbuja

	Ietraciones				
Posición					
0	5	1	1	1	1
1	3	3	2	2	2
2	4	4	4	3	3
3	2	2	3	4	4
4	1	5	5	5	5

N=5

Vector: num

Burbuja tradicional: Se debe comparar cada elemento con los siguientes elementos

Ciclo i	inicia en el primer elemento hasta penúltimo	
Ciclo j	inicia en i+1 (siguiente de i) hasta último	

Ciclos anidados	0	1	2	3
	1,2,3,4	2,3,4	3, 4	4

Condicional=> Comparar la num[i] con el num[j]

Estructuras de Datos

Ordenamiento iterativo – Método Burbuja

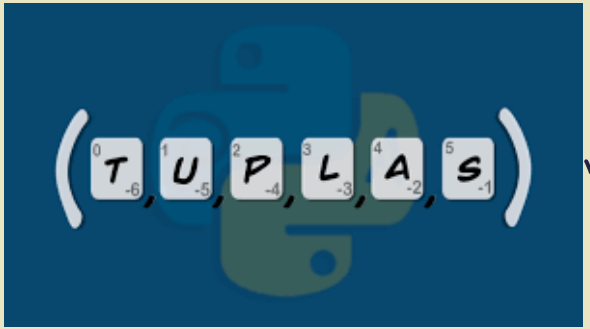
```
#Funciones
def valida_entero(etiqueta):

    while True:
          try:
              dato = int(input(etiqueta))
              break
          except ValueError:
                    print(etiqueta, "  debe ser entero  Intenta de nuevo...")
    return dato

def ordenamiento_burbuja(numeros):
    for i in range (0,N-1):
        for j in range (i+1,N):
            if numeros[i]>numeros[j]:
                t=numeros[i]
                numeros[i]=numeros[j]
                numeros[j]=t
    return numeros
```

```
#Programa principal
N=valida_entero("Ingrese cantidad de elementos: ")
numeros=[]
for i in range(N):
    num=valida_entero("Número: ")
    numeros.append(num)
print( "Lista Numeros: ",numeros)
#Llamado a la función
numeros=ordenamiento_burbuja(numeros)
print ("Lista ordenada: ",numeros)
```

Estructuras de Datos

TÚPLAS

Las tuplas son estructuras que, una vez creadas o definidas, NO permiten modificarse a lo largo de la ejecución de un programa, lo cual les da la caracterización de ser **estructuras inmutables**.



Estructuras de Datos

TÚPLAS

```
>>> tupla=(1,"Juan",5,"Pedro",10.5,[2-3],"Sergio")
>>> tupla
(1, 'Juan', 5, 'Pedro', 10.5, [-1], 'Sergio')
>>> tupla[1]
'Juan'
>>> tupla.append(15)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    tupla.append(15)
AttributeError: 'tuple' object has no attribute 'append'
>>> tupla.extend([30,40])
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    tupla.extend([30,40])
AttributeError: 'tuple' object has no attribute 'extend'
>>> tupla.remove(5)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    tupla.remove(5)
AttributeError: 'tuple' object has no attribute 'remove'
```

Estructuras de Datos

TÚPLAS

Concatenación

La concatenación es la combinación de tuplas. Digamos que tenemos las siguientes tuplas:

```
tuple1 = (1,2,3,4,5)

tuple2 = (6,7,8,9,10)
```

Si queremos concatenar la tuple1 con la tuple2, simplemente debemos escribir:

```
tup = tuple1 + tuple2
```

Fíjate en que he usado el operador + para realizar la concatenación. Esto derivará en la siguiente salida:

```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Estructuras de Datos

TÚPLAS

Repetición

La repetición de tuplas se puede llevar a cabo mediante el uso del operador *. Si quieres repetir el contenido de la tupla tres veces, simplemente haz lo siguiente:

```
tuple1 * 3
```

El resultado de esta sentencia será:

```
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

Estructuras de Datos

TÚPLAS

Comprobar elementos

Para comprobar si un elemento existe en la tupla, simplemente tenemos que utilizar la palabra clave in de este modo:

```
7 in tuple1
```

Esto devolverá un false como una casa, ya que el valor 7 no se encuentra dentro de tuple1.

Estructuras de Datos

TÚPLAS

Buscar

Si queremos saber en qué índice está localizado un valor concreto, podemos hacerlo. Solo hay que usar index y nos devolverá el índice donde se encuentra. Por ejemplo, si queremos encontrar la ubicación del elemento 5 en la tuple1, haremos lo siguiente:

```
tuple1.index(5)
```

Este caso, nos devolverá el valor 4, que es la localización del elemento 5.

Estructuras de Datos

TÚPLAS

Contador

Una acción que se suele repetir mucho cuando se trabaja con tuplas es la de contar el número de veces que un elemento existe en la tupla. Digamos que tenemos la siguiente tupla:

```
tuple3 = (65,67,5,67,34,76,67,231,98,67)
```

Si queremos saber cuántas veces existe el elemento 67 en la tuple3, haremos lo siguiente:

```
tuple3.count(67)
```

El resultado de esta sentencia debería ser 4. Las cuatro veces que se repite el elemento 67.

Estructuras de Datos

TÚPLAS

Indexación

La indexación es el proceso de acceder al elemento de una tupla mediante un índice. Si queremos acceder al quinto índice de la tuple3, haremos lo siguiente:

```
tuple3[4]
```

Esto nos devolverá 34.

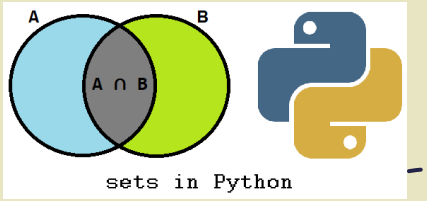
Un índice también puede ser negativo, si el recuento lo iniciamos desde la derecha de la tupla. Por lo tanto, el resultado de tuples3[-6] será 34. ¿Qué pasaría si hacemos referencia a un índice fuera de rango? Es decir, ¿qué pasa si declaramos tuples3[-11]? Esto es lo que pasaría en ese caso:

```
Traceback (most recent call last):
  File "tuple.py", line 2, in <module>
    print tuple3[-11]
IndexError: tuple index out of range
```

Recuerda que los índices negativos empiezan desde -1. Por lo tanto, si escribes el índice -0, es lo mismo que indicar 0. Es por eso que, tuples3[-0], devolverá 65.

Estructuras de Datos

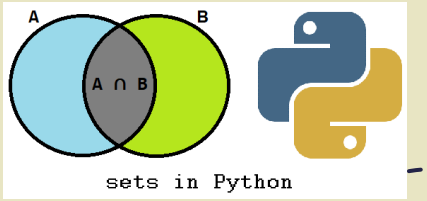
CONJUNTOS



Un **conjunto** es una colección no ordenada de objetos únicos. Python provee este tipo de datos «por defecto» al igual que otras colecciones más convencionales como las listas, tuplas y diccionarios.

Los conjuntos son ampliamente utilizados en lógica y matemática, y desde el lenguaje podemos sacar provecho de sus propiedades para crear código más eficiente y legible en menos tiempo.

Estructuras de Datos



CONJUNTOS

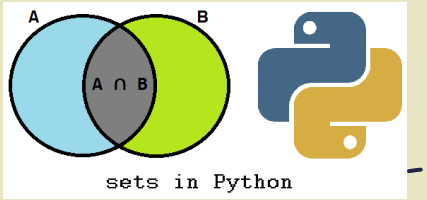
Un conjunto se crea colocando todos los elementos (elementos) entre llaves {}, separados por comas o usando la función incorporada set(). Puede tener cualquier número de elementos y pueden ser de diferentes tipos (entero, flotante, tupla, cadena, etc.). Sin embargo, un conjunto no puede contener elementos mutables como listas, conjuntos o diccionarios.

Conjunto = {12, 22, 33}

Es importante recordar que, los elementos de un conjunto no tiene un orden predefinido como las listas o las tuplas; por lo tanto, las posiciones de los elementos no importan y pueden variar entre la definición y la impresión del conjunto.

Estructuras de Datos

CONJUNTOS



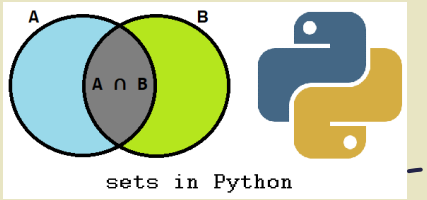

```
>>> conjunto1={1,2,4,"Pedro","Luis"}
>>> conjunto1
{1, 2, 4, 'Luis', 'Pedro'}
>>> conjunto2=set([2,5,6,"Mara","Luis"])
>>> conjunto2
{2, 5, 6, 'Luis', 'Mara'}
>>> conjunto1.add("Hugo")
>>> conjunto1
{1, 2, 4, 'Hugo', 'Luis', 'Pedro'}
>>> conjunto1.remove(2)
>>> conjunto1
{1, 4, 'Hugo', 'Luis', 'Pedro'}
>>>
```

Add: Adicionar elementos

Remove: Eliminar elementos

Estructuras de Datos

CONJUNTOS




```
>>> A={1,3,5,7,9,12}
>>> B={5,7,9,15,20,30}
>>> C=A | B
>>> C
{1, 3, 5, 7, 9, 12, 15, 20, 30}
>>> D=A & B
>>> D
{9, 5, 7}
>>> E=A - B
>>> E
{1, 3, 12}
>>> F=B - A
>>> F
{20, 30, 15}
```

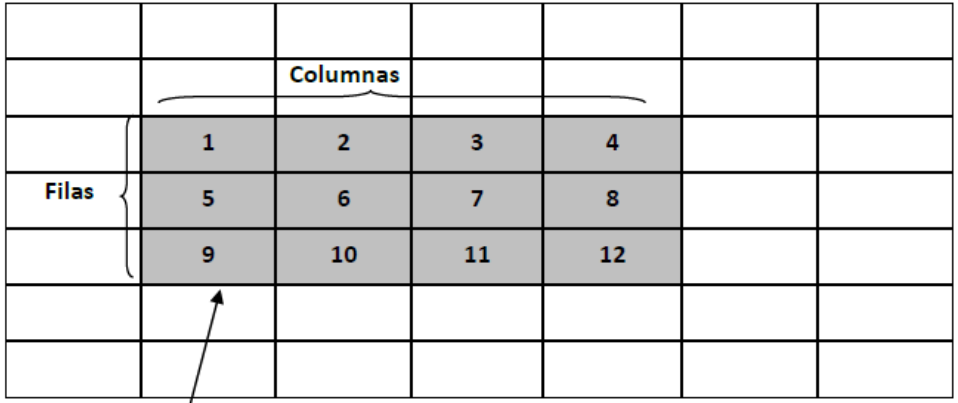
Operaciones Conjuntos

- **Unión: |**
- **Intersección: &**
- **Diferencia: -**

Estructuras de Datos

Matrices – Listas dentro de Listas

MEMORIA



Matriz de Números: En esta matriz se almacenan números, que serán utilizados en el proceso. En este caso, la matriz maneja la dimensión horizontal ó fila y la dimensión vertical o columna.
Número de Filas: 3
Número de Columnas: 4

Estructuras de Datos

Matrices – Listas dentro de Listas

1	2	3
4	5	6

Matriz

Filas y Columnas empiezan en cero

Fila 0=> 1,2,3

Fila 1=>4,5,6

Para referenciar un elemento de la matriz:

1=matriz[0][0]

Matriz[i][j] para referenciar un elemento de la matriz. El que se encuentra en la fila i, columna j

Estructuras de Datos

Matrices – Listas dentro de Listas

Columnas

Filas

$$\begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix} = \begin{bmatrix} 2,4,6 \\ 8,10,12 \\ 14,16,18 \end{bmatrix} = [[2,4,6],[8,10,12],[14,16,18]]$$

MATRIZ
3x3
mxn

3 LISTAS

En Python

matriz = [[2,4,6], [8,10,12], [14,16,18]]

m = Filas

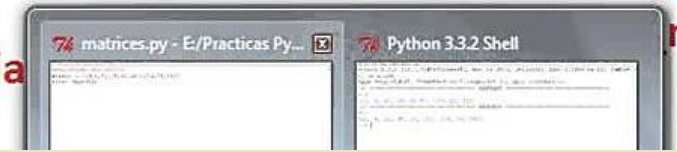
n = Columnas

Si quiero hacer referencia a x elemento de la matriz:

matriz = [m][n]

Haciendo referencia

matriz [1][2]



Estructuras de Datos

Matrices – Listas dentro de Listas

```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bi
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> matriz=[[1,2,3],[4,5,6]]
>>> matriz
[[1, 2, 3], [4, 5, 6]]
>>> matriz[0]
[1, 2, 3]
>>> matriz[1]
[4, 5, 6]
>>> matriz[0][0]
1
>>> matriz[1][1]
5
>>>
```

Estructuras de Datos

Matrices – Listas dentro de Listas

Para el recorrido de una matriz (lista dentro de una lista), es decir pasar por cada uno de sus elementos, se utilizan DOS ciclos FOR anidados, uno dentro de otro, uno para las filas y otro para las columnas

Matriz Letras	a	t	i	Fila 0
	f	o	j	Fila 1
	u	r	e	Fila 2
	Col 0	Col 1	Col 2	

Recorrer la matriz: Pasar por cada uno de los elementos=> 2 ciclos anidados

Ciclo fila	0	1	2
Ciclo Columna	0,1,2	0,1,2	0,1,2

Estructuras de Datos

Matrices – Listas dentro de Listas

Ejemplo

A continuación se presenta el Informe Semanal de las ventas diarias realizadas por la Compañía **SweetCO** de sus cinco (5) principales productos.

VENTAS DE LA SEMANA

Producto	L	M	W	J	V	S	D
1	100	88	92	94	85	110	118
2	30	42	31	32	38	40	37
3	23	35	39	45	55	60	61
4	45	50	56	65	47	57	68
5	18	25	33	21	22	28	32

PRECIOS DE PRODUCTOS

1	2	3	4	5
1500	5000	6500	2500	22500

Usted ha sido contratado para desarrollar un programa que reciba la matriz de Ventas Semanales y calcule los ingresos de la compañía a partir del vector de precios de sus Productos.

Responda las siguientes preguntas del negocio:

- Producto que genera mas ingresos en la semana
- El día de la semana con mayor ingresos por ventas

Estructuras de Datos

Matrices – Listas dentro de Listas

Ejemplo - Código

```
#Funciones
def prodMayVtasSem(mat):
    vsumprod = []
    for f in range(len(mat)):
        sum = 0
        for c in range(len(mat[f])):
            sum += mat[f][c]
        vsumprod.append(sum)

    print(vsumprod)
    return vsumprod.index(max(vsumprod)) + 1
```

```
# Programa Principal
mventas = [[100, 88, 92, 94, 85, 110, 118],
           [30, 42, 31, 32, 38, 40, 37],
           [23, 35, 39, 45, 55, 60, 61],
           [45, 50, 56, 65, 45, 57, 68],
           [18, 25, 33, 21, 22, 28, 32]]

pmyVtasSem = prodMayVtasSem2(mventas)
print("El producto que mas vende a la semana
es:", pmyVtasSem)
```

Estructuras de Datos

Diccionarios

Los diccionarios constituyen un tipo de estructura de datos en la que, sus elementos, se encuentran compuestos por pares "**clave:valor**", en donde la clave es un valor de cualquier tipo y lleva asociado otro valor (que, a su vez, también puede ser de cualquier tipo, sin necesidad, tampoco, que sea del mismo tipo que la clave).

Dichos pares "**clave**" y "**valor**" irán separados por dos puntos ("**:**").

Los elementos Clave-valor de un diccionario se separan por comas y se encierran entre llaves **{}**.

Estructuras de Datos

Diccionarios

Los diccionarios permiten almacenar **valores** indexados, a través de **claves,** lo cual permite realizar una búsqueda más eficiente

Ejemplos:

capitales = {“Colombia”:”Bogotá”, “Perú”:”Lima”, “Argentina”:”Buenosaires”}

empleado = {'nombre':"Sergio", 'apellido':"Medina", 'edad':57, 'salario':3500000}

Estructuras de Datos

Diccionarios -Acceso

Una vez que almacenamos los datos en el diccionario, vamos a acceder a ellos

Con el **método get()** de un diccionario, podemos obtener el valor de una clave, pero si no existe la clave devolver un mensaje en string como respuesta.

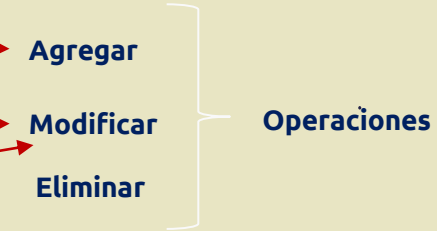
```
>>> empleado={'nombre':"Sergio Medina",'cargo':"Programador",'salario':4000000}
>>> empleado
{'nombre': 'Sergio Medina', 'cargo': 'Programador', 'salario': 4000000}
>>> empleado['nombre']
'Sergio Medina'
>>> empleado.get('nombre')
'Sergio Medina'
>>> empleado['email']
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    empleado['email']
KeyError: 'email'
>>> empleado.get('email',"NO ENCONTRADO")
'NO ENCONTRADO'
```

Cuando la clave NO es encontrada en el diccionario

Estructuras de Datos

Diccionarios - Operaciones

```
>>> articulos={1:"Lapiz",2:"Borrador",3:"Cuadernos"}
>>> articulos
{1: 'Lapiz', 2: 'Borrador', 3: 'Cuadernos'}
>>> articulos[4]="Calcualdora"
>>> articulos
{1: 'Lapiz', 2: 'Borrador', 3: 'Cuadernos', 4: 'Calcualdora'}
>>> articulos[4]="Calculadora"
>>> articulos
{1: 'Lapiz', 2: 'Borrador', 3: 'Cuadernos', 4: 'Calculadora'}
>>> articulos[5]="Refresco"
>>> articulos
{1: 'Lapiz', 2: 'Borrador', 3: 'Cuadernos', 4: 'Calculadora', 5: 'Refresco'}
>>> del articulos[5]
>>> articulos
{1: 'Lapiz', 2: 'Borrador', 3: 'Cuadernos', 4: 'Calculadora'}
```



Estructuras de Datos

Diccionarios

$dic1 = \{\ 10: 7, 20: (1,2,3), 30: [\text{'Control'}, \text{'}E\text{ducación'}]\ \}$

7	(1, 2, 3)	['Control', 'Educación']
10	20	30

$tienda = \{$ 'item' : ['Lápiz', 'Carpeta', 'Marcador'],
'cantidad' : [3, 10, 5],
'valor' : [3.50, 4.25, 7.85] $\}$

item	cantidad	valor
'Lápiz'	3	3.50
'Carpeta'	10	4.25
'Marcador'	5	7.85

Estructuras de Datos

Diccionarios - métodos

clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Estructuras de Datos

Diccionarios - métodos

clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Estructuras de Datos

Diccionarios – setdefault

El setdefault()método devuelve el valor del elemento con la clave especificada.

Si la clave no existe, inserte la clave, con el valor especificado, vea el ejemplo a continuación

Estructuras de Datos

Diccionarios – set default

Sintaxis:

```
dictionary.setdefault(keyname, value)
```

Keyname: *Requerido. El nombre clave del elemento del que desea devolver el valor.*

Value: *Opcional.*

- *Si la clave **existe**, este parámetro no tiene efecto.*
- *Si la clave **no existe**, este valor se convierte en el valor de la clave.*
- *Valor por defecto **None**.*

Estructuras de Datos

Diccionarios – set default

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.setdefault("color", "white")

print(x)
```

Estructuras de Datos

Diccionarios -Ejercicio

La institución educativa “SamEduca” cuenta con N docentes, conociendo de cada uno de ellos su número de cédula, nombre, categoría y números de horas laboradas en el mes. Se pide realizar un programa que calcule el valor de los honorarios de cada docente y el valor total a pagar por concepto de honorarios. Para este proceso, nos suministran el diccionario donde se define el valor de la hora para cada categoría, así:

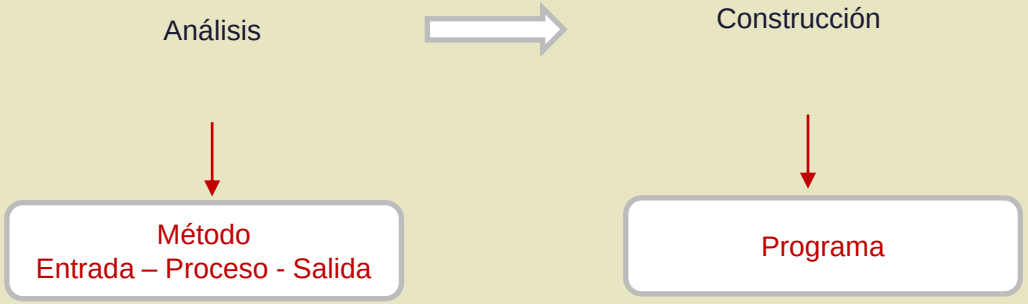
diccionario_categoria={1:25000,2:30000,3:40000,4:45000,5:60000}

Se debe imprimir el nombre del docente, el valor de sus honorarios y el valor total de honorarios, el de los N docentes.

Estructuras de Datos

Diccionarios -Ejercicio

Metodología -> Pensamiento lógico estructurado



Estructuras de Datos

Diccionarios -Ejercicio

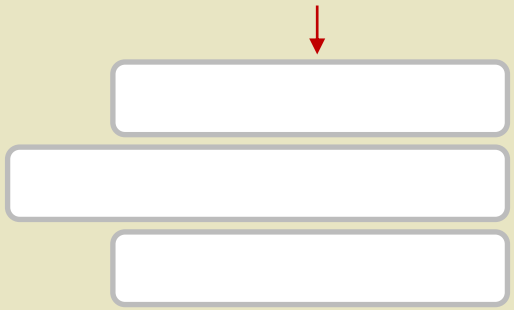
1 2 3 4 5 6 7

Análisis –> Ejercicio Listas

Entrada LEER

Proceso

Salida IMPRIMIR



Estructuras de Datos

Diccionarios –Ejercicio – Versión 1

```
# Programa para manejo de diccionarios
# Autor: Sergio Medina
# Fecha: 14/06/2022

diccionario_categoria={1:25000,2:30000,3:40000,4:45000,5:60000}
N=int(input("Cantidad de docentes: "))
total_honorarios=0
for i in range(N):
    cedula=int(input("Cédula docente: "))
    nombre=input("Nombre docentes: ")
    categoria=int(input("Categoría: "))
    horas=int(input("Horas laboradas en mes: "))
    honorarios=horas*diccionario_categoria.get(categoria)
    total_honorarios+=honorarios
    print("Nombre docente: ",nombre)
    print("Honorarios: ",honorarios)
print("Total Honorarios: ",total_honorarios)
```

Estructuras de Datos

Diccionarios –Ejercicio – Versión 2 (Validación)

```
# Programa para manejo de diccionarios
# Autor: Sergio Medina
# Fecha: 14/06/2022

diccionario_categoria={1:25000,2:30000,3:40000,4:45000,5:60000}
N=int(input("Cantidad de docentes: "))
total_honorarios=0
for i in range(N):
    cedula=int(input("Cédula docente: "))
    nombre=input("Nombre docentes: ")

    while True:
        try:
            categoria=int(input("Categoría: "))
            if diccionario_categoria.get(categoria,"ERROR")=="ERROR":
                print("Categoría NO Existe")
                continue
            break
        except ValueError:
            print("La categoría debe ser un dato entero")

    horas=int(input("Horas laboradas en mes: "))
    honorarios=horas*diccionario_categoria.get(categoria)
    total_honorarios+=honorarios
    print("Nombre docente: ",nombre)
    print("Honorarios: ","{:,.2f}".format(honorarios))
print("Total Honorarios: ","{:,.2f}".format(total_honorarios))
```

Talleres:

Realizar los ejercicios usando arreglos y colecciones

Estructura de Datos

Situación Problema: Pares e Impares

Se desea realizar un programa en el cual se ingresen números enteros, los cuales se deben almacenar en una lista. Se debe ingresar números hasta que el número ingresado sea 99999. Una vez creada la lista, se desea conocer cuales y cuántos son pares e impares.

Estructura de Datos

Situación Problema: Contar palabras

Dada una lista con nombres completos de personas, realizar un programa que genere una segunda con la cantidad de palabras de cada uno de los nombres. La lista de nombres debe llenarse a través de nombres que se ingresan por teclado, hasta que el nombre ingresado sea “FIN”

Se debe imprimir la lista de nombres y la lista con la cantidad de palabras de cada nombre.

Estructura de Datos

Situación Problema: Calificación tripulante

Dada la siguiente información sobre las calificaciones de estudiantes de una institución educativa:

- Código
- Nombre
- Nota 1 (Peso de 30%)
- Nota 2 (Peso de 30%)
- Nota 3 (Peso de 40%)

El proceso se termina cuando el código que se ingrese sea 999.(Bandera)
Se pide calcular:

La nota definitiva de cada estudiante e indicar con un mensaje si aprobó o reprobó, utilizando funciones
Para aprobar, la nota deber ser mayor o igual a 3.0 y la información en su totalidad se debe almacenar en listas

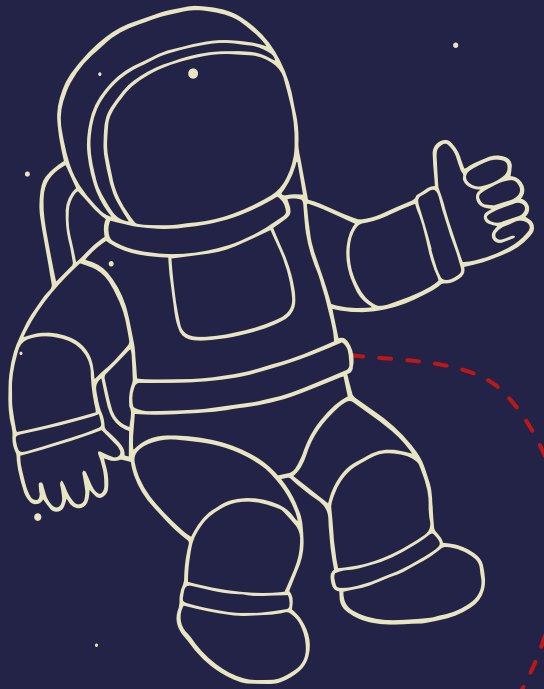
Estructura de Datos

Se realiza la compra de N artículos, en donde se ingresa el código del artículo y la cantidad y mediante el uso de diccionarios para los nombres y valores unitarios de los artículos, el programa debe obtener el nombre de cada artículo, cantidad comprada, valor unitario, valor total de acuerdo a la cantidad comprada y finalmente calcular el valor total de la compra.

Se suministra el diccionario de nombres de artículo y otro con los valores unitarios.

articulos={1:"Lapiz",2:"Cuadernos",3:"Borrador",4:"Calculadora",5:"Escuadra"}

valores={1:2500,2:3800,3:1200,4:35000,5:3700}



Programa académico CAMPUS

Ciclo 1: Fundamentos de Programación