

Clase 1 - Parte 2: Programación

Introducción a la Programación.

Temario

Estructuras de control:

- Iterativas:

- .while

- . for

- . Uso de break, continue y exit

Estructuras de datos:

- Listas

- diccionarios

- conjuntos



Estructuras de Control Repetición

Iteraciones

- **Bucles:** Permiten **ejecutar cierto código un número reiterado de veces** hasta que se cumpla una condición.
- Python tiene dos sentencias iterativas:

```
while
```

```
for .. in
```

Iteramos con for

Forma general:

```
for variable in lista de Valores:  
    sentencias
```

La variable toma **todos los elementos que aparecen en la lista de valores** y luego termina la iteración.

Iteramos con for

- **Función range():** Devuelve una lista de números enteros.

Formas de usarla:

- 1 argumento:

for i in range(5):, devuelve [0,1,2,3,4] - Desde 0 hasta el argumento -1

- 2 argumentos:

for k in range(2,5):, devuelve [2,3,4] - Desde el arg.1 hasta el 2do arg. -1

- 3 argumentos:

for j in range(2,5,2):, devuelve [2,4] - Desde el arg.1 hasta el 2do arg. -1, pero con un incremento de 2

Sentencia while

Forma general:

While **condición:**
sentencias

- La condición se evalúa en cada ejecución del bucle y mientras sea verdadera, la iteración continúa.

Importante: La condición DEBE hacerse falsa en algún momento, ¿Qué pasa si esto no sucede?

Sentencia while

```
print ("ingrese un número. Para finalizar 0")
num=int(input())
suma=0
while (num !=0):
    suma=suma + num
    print ("ingrese un número, 0 finaliza")
    num=int(input())

print("la suma es:", suma)
```


While vs for

- Ambas son **sentencias iterativas**

En ambas sentencias, **las líneas pertenecientes al bucle** deben estar **indentadas**

Diferencia:

- La **sentencia while evalúa una condición** que debemos asegurarnos se haga **falsa en algún momento**
- La **sentencia for, itera un número fijo de veces**: hasta que la variable tome todos los posibles valores de la lista.

Uso del Break – Continue - Exit

- **Ejemplo de validación de datos de entrada:**

while True:

- `name=input('ingrese nombre')`

`if name == 'pepe':`

`break`

`print ('Hola',name,'bienvenido al sistema')`

Uso del Break – Continue - Exit

```
• while True:
•     name=input('ingrese nombre')
•     if name != 'pepe':
•         continue          #vuelve a repetir el ingreso de datos y
                             #saltea el resto del cuerpo del while
•     pwd=input('ingrese contraseña') # si nombre es válido
•     if pwd == 'cleopatra':
•         break  #sale del while
•     ('Hola',name,'ingreso válido')
```

Uso del Break – Continue - Exit

```
import sys #proporciona acceso a algunas variables
           # y funciones que interactúan con el
           #interprete de python
```

```
.....
```

```
while True:
```

```
    res=input('ingrese un nombre, exit para salir')
```

```
    if res == 'exit':
```

```
        sys.exit
```

```
    print ('Ud. escribió',res)
```

```
.....
```

La función **exit** **detiene la ejecución** del programa y **cierra** el intérprete de **Python**.

break, continue y exit se pueden usar tanto con while como con for



Estructuras de datos.

Colecciones:

- Listas**

- Diccionarios**

- Conjuntos**

Listas

- **Colección ordenada de datos heterogéneos.**
De tamaño variable.
- Puede contener **cualquier tipo de datos**, inclusive listas.
- Ejemplos:
 - **Lista1=[]**
 - Lista2=[1,2,3]**
 - Lista3=[1, "Hola"]**
 - Lista4= [22, True, 'una lista', [1,7]]**

Listas

¿Cómo accedemos a los elementos de la lista?

- A través del índice del elemento (**posición dentro de la lista**), expresado entre corchetes **[]**.

IMPORTANTE: los índices comienzan en **0**

Ejemplo:

```
print (Lista2[2])
```

- **Lista4[1] = False** → esto provoca que el **2do** elemento de la lista se cambie.

Listas

Para **acceder** a elementos de tipo “listas”, se debe usar también **[]**.

- Ej. Para la lista **Lista4=[22, True, 'una lista', [1,7]]**
- El **primer corchete** indica **posición de la lista exterior**, los **otros** indican **posición de las listas interiores**.
- Ej.: **Lista4[3][1]**, devuelve **7**
- Se pueden usar **índices negativos**. En ese caso se comienza a contar desde atrás.
- Ej.: **Lista4[-3]**, devuelve **True**

Listas

```
lis1 = [22, True, 'una lista', [1,7]]
```

	Descripción	Ejemplo	Resultado
append	Agrega un elemento al final de la lista	<code>lis1.append(4)</code>	<code>[22, True, 'una lista', [1,7], 4]</code>
count	Cuenta el número de apariciones de un elemento de la lista	<code>lis1.count(22)</code>	1
index	Devuelve la posición de un elemento dentro de la lista	<code>lis1.index('una lista')</code>	2
del	Elimina un elemento	<code>del lis1[2]</code>	<code>[22, True, [1,7]]</code>

Listas

```
lis1= [22, True, 'una lista', [1,7]]
```

verifica pertenencia : **True in lis1**

```
lis1.insert(1,24) # lis1=[22,24,True,'una lista',[1,7]]
```

```
lis1.remove('una lista') #si no existe el elemento da error
```

```
lis1.sort() # debe ser homogénea. Modifica lis1
```

```
lis1.sort(reverse=True) # ordena en sentido inverso.
```

Listas

Slicing:

- Permite seleccionar **porciones** de listas:

— Para seleccionar parte de una lista se debe colocar **inicio:fin**. Indica que queremos la parte de la lista que comprende desde el elemento inicio hasta el elemento anterior a fin. **NO** incluye al elemento cuyo índice es fin.

```
lis1 = [22, True, 'una lista', [1,7]]
```

Ej.: `lis1[1:3]`, devuelve la lista `[True, 'una lista']`

Listas

Slicing:

Si no se indica el inicio o fin, se toman por defecto las posiciones de inicio o fin de la lista.

Ej.: `lis1[:2]`, devuelve la lista `[22,True]`

`lis1[2:]`, devuelve la lista `['una lista',[1,7]]`

```
>>> long=5
>>> cadena='lista'
>>> lista=['esto es una',cadena,'de', long, 'elementos']
>>> print (lista)
['esto es una', 'lista', 'de', 5, 'elementos']
>>> print (lista[0])
esto es una
>>> print (lista[1:-2])
['lista', 'de']
>>> print (lista[2:-4])
[]
>>>
```

Listas

Las **variables** de tipo **lista** contienen un **puntero** a la **colección de datos**.

Si **2 variables** **apuntan a la misma lista**, cuando **se modifica** a través de cualquiera de dichas variables, **se refleja en la lista**:

```
lis1= [22, True, 'una lista', [1,7]]
```

```
lis2=lis1
```

```
print (lis1[1]) # muestra True
```

```
print (lis2[1] ) # muestra True
```

```
lis2[2]= 45 # lis1= [22, True, 45, [1,7]]
```

```
print (lis1[2] ) #muestra 45
```

Diccionarios

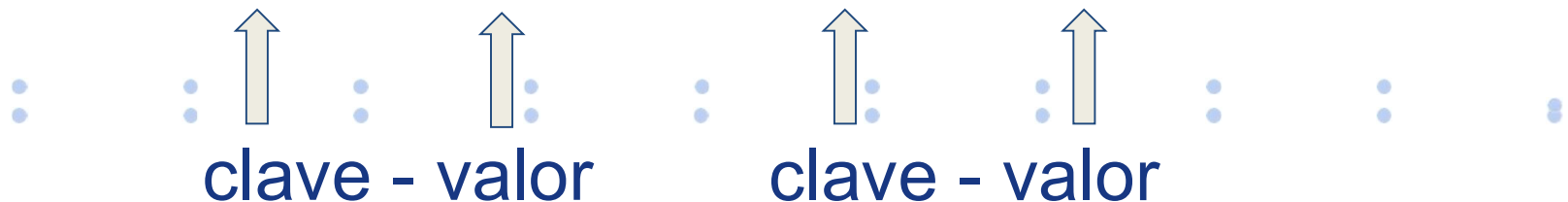
Los diccionarios son un tipo de **estructuras de datos** que **permite guardar un conjunto no ordenado de pares clave-valor**, siendo las **claves únicas** dentro de un mismo diccionario.

Para crear un diccionario vacío usamos:

`dic1={}` o `dic1=dict()`

Para crearlo inicializado:

`dic1={21345:'Perez Juan', 23456:'Roni Ana'}`


clave - valor clave - valor

Diccionarios

Devuelve el número de elementos que tiene el diccionario
`len(dic1)`

Devuelve una lista con las claves del diccionario
`dic1.keys()`

Devuelve una lista con los valores del diccionario
`dic1.values()`

Devuelve el valor del elemento con clave key.

`dic1.get(21345)`

`dic1[21345]` #lo accede por la clave escrita con []

Diccionarios

Inserta un elemento en el diccionario clave:valor.

Si la clave existe no lo inserta

`dic1.setdefault(125478,'Armest Dani')`

Insertamos un elemento en el diccionario con su clave:valor

`dic1[96523] = 'Diaz Lucas'` # si existe la clave, cambia el valor

Elimina todos los elementos de un diccionario

`dic1.clear()`

Diccionarios

Devuelve los elementos clave –valor como lista de tuplas

```
tuplas=dic1.items()
```

Recorre un diccionario, imprimiendo su clave-valor

```
for k,v in dic1.items():
```

```
    print ( k,'-->',v)
```

#Recorre un diccionario, imprimiendo su clave

```
for k in dic1:
```

```
    print ( k)
```

#Recorre un diccionario, imprimiendo su valor

```
for k in dic1:
```

```
    print ( dic1[v])
```

Diccionarios

```
# Elimina una par clave- valor  
dic1.pop(21345)  
del dic1[21345]
```

Sets - Conjuntos

Python también incluye un tipo de dato para *conjuntos*.

Un conjunto es una **colección no ordenada de elementos heterogéneos no repetidos**. No se **acceden** por índice sino **en forma aleatoria**.

Los **usos** básicos de éstos incluyen **verificación de pertenencia, eliminación de entradas duplicadas** y distintas ramas de la matemática.

Los conjuntos también **soportan operaciones** como la unión, intersección, diferencia, y diferencia simétrica.

Sets - Conjuntos

Para crear un conjunto vacío

```
conj= set()
```

#Para agregar elementos

```
conj.add(2)
```

#Para eliminar elementos

```
conj.remove(2) # si no existe da error
```

```
conj.discard(2) #si no existe no hace nada
```

Cantidad de elementos del conjunto

```
len(conj)
```

Sets - Conjuntos

#Pertenencia

3 in conj

4 not in conj

#Para crear un conjunto inicializado se usan llaves

```
>>> canasta = {'manzana', 'naranja', 'manzana',  
'pera', 'naranja', 'banana'}
```

```
>>> canasta={'manzana','naranja','pera','banana'}
```

#Crea el conjunto pero no pone los repetidos

#Eliminar todos los elementos del conjunto

```
conj.clear()
```

Sets - Conjuntos

conversión

```
>>> list({1, 2, 3, 4})
```

```
[1, 2, 3, 4]
```

#conjunto a lista

```
>>> set([1, 2, 2, 3, 4])
```

```
{1, 2, 3, 4}
```

#lista a conjunto

Ejercicio 1

Implemente un programa que pida al usuario que ingrese por teclado una serie de 10 números y que luego imprima los números ingresados que resultan mayores que el promedio. (usar listas).

Ejecutar en colab

```
lista_numeros=[]  
suma=0
```

```
• • • • •  
for i in range(10):
```

```
•   numero = int(input("ingrese un numero: "))
```

```
•   lista_numeros.append(numero)
```

```
•   suma = suma + numero
```

```
•  
promedio = suma / len(lista_numeros)
```

```
• print("el promedio es: ",promedio)
```

```
print("*****")
```

```
print("los valores mayores al promedio son: ")
```

```
for i in lista_numeros:
```

```
    if i > promedio:
```

```
        print(i)
```


Ejercicio 2

Implemente un programa que pida al usuario que ingrese por teclado el DNI y el promedio de una serie de alumnos. Luego, que guarde la información en un diccionario y determine quién tiene el mejor promedio.

Ejecutar en colab

```
• lista_alumnos = dict()
• confirmacion = input("ingresa alumno?? si/no: ")
• while confirmacion == "si":
•     dni = input("ingrese el dni del alumno: ")
•     promedio = float(input("ingrese promedio del alumno: "))
•     lista_alumnos[dni] = promedio
•     confirmacion = input("ingresa otro alumno??? si/no: ")

• print(lista_alumnos)

• lista_promedios = list(lista_alumnos.values())
• lista_promedios.sort() #ordena de menor valor a mayor. ascendente
• print("el mejor promedio es: ", lista_promedios[-1])

• for c,v in lista_alumnos.items():
•     if v==lista_promedios[-1]:
•         print("el alumno con mayor promedio tiene DNI: ", c)
•         break
```

