Unidad 3

Implementación de algoritmos en Python.

Ejercicios

1. Hacer un algoritmo que, dados dos valores numéricos A y B, escriba un mensaje diciendo si A es mayor, menor o igual a B.

```
print("Este algoritmo, luego de ingresar los valores de A y B,\nimprimira si A es mayor, menor o igual a B")
print ("\n")
a = input ("Ingrese el valor de A: ")
b = input ("Ingrese el valor de B: ")
if a > b :
    print("A es mayor (>) que B")
elif a < b :
    print("A es menor (<) que B")
else:
    print ("A es igual (=) a B")
```

Ejercicios

2. Diseñe un algoritmo que permita obtener el promedio de 5 notas de un curso, valoradas de 0 a 5 y ponderadas con un 30, 15, 15, 20, 20%, y escriba aprobado si la calificación es mayor o igual a 3. Finalmente, independiente de si ganó o perdió, muestre el promedio.

Ejercicios

```
print('Este programa le calculara la nota final de una materia, luego de\n ingresar las 5 notas del semestre')
n1 = input("Ingrese la nota 1 (30%): ")
if n1<0 or n1>5:
    print("Recuerde que las notas deben estar entre 0 y 5. Intente de nuevo por favor")
    n1 = input("Ingrese la nota 1 (30%): ")
n2 = input("Ingrese la nota 2 (15%): ")
if n2<0 or n2>5:
    print("Recuerde que las notas deben estar entre 0 y 5. Intente de nuevo por favor")
    n2 = input("Ingrese la nota 2 (15%): ")
n3 = input("Ingrese la nota 3 (15%): ")
if n3<0 or n3>5:
    print("Recuerde que las notas deben estar entre 0 y 5. Intente de nuevo por favor")
    n3 = input("Ingrese la nota 3 (15%): ")
n4 = input("Ingrese la nota 4 (20%): ")
if n4<0 or n4>5:
    print("Recuerde que las notas deben estar entre 0 y 5. Intente de nuevo por favor")
    n4 = input("Ingrese la nota 4 (20%): ")
n5 = input("Ingrese la nota 5 (20%): ")
if n5<0 or n5>5:
    print("Recuerde que las notas deben estar entre 0 y 5. Intente de nuevo por favor")
    n5 = input("Ingrese la nota 5 (20%): ")

notaf = (n1*0.3 + n2*0.15 + n3*0.15 + n4*0.2 + n5*0.2)

if notaf >= 3:
    print ("Su nota final fue: "+ str(notaf) + ". APROBADO")
else:
    print ("Su nota final fue: "+ str(notaf) + " - REPROBADO").
```

Bucles while

Las sentencias dentro de la sentencia **while** son ejecutadas repetidas veces, siempre y cuando la condición se mantenga. En el momento que se evaluada como **False**, la siguiente sección será ejecutada.

```
i = 1
while i <= 5:
        print(i)
        i = i + 1

print(“Fin de programa”)
```

```
>>>
1
2
3
4
5
Fin de programa
>>>
```

Bucles while

¿Cuántos número imprime este código?.

```
i = 3
while i >= 0:
    print(i)
    i = i -1
```

4

Bucles while

El **bucle infinito** es un tipo especial de bucle while; nunca deja de ser ejecutado. Su condición permanece siempre en **True**.

```
while 1 == 1:
    print ("Entró en un ciclo infinito!!!")
```

Para parar el ciclo infinito, presione **Ctrl + C** o cerrar la ventana.

break

Para finalizar un bucle **while** prematuramente, se puede utilizar la sentencia **break**.
Cuando se está dentro de un bucle, la sentencia **break** hace que este finalice prematuramete.

```
i = 0
while 1 == 1:
    print (i)
    i = i + 1
    if i >= 5:
        print("Breaking")
        break

print("Fin de programa")
```

```
>>> ====================
>>>
0
1
2
3
4
Breaking
Fin de programa
```

break

¿Cuántos número imprime este código?.

```
i = 5
while True:
    print(i)
    i = i -1
    if i <= 2:
        break
```

3

continue

Otra sentencia que puede ser utilizada dentro de los bucles es **continue**.

A diferencia de break, **continue** vuelve al principio del bucle en vez de detenerlo.

```
i = 0
while True:
        i = i +1
        if i == 2:
                print (“Paso 2”)
                continue
        if i == 5:
                print (“Braking”)
                break
        print(i)
print(“Fin de programa”)
```

```
>>>
1
Paso 2
3
4
Braking
Fin de programa
>>>
```

Listas

Lista son otro tipo de objeto en Python. Son utilizadas para almacenar una lista indexada de objetos.

Una lista se crea utilizando corchetes con comas separando a los objetos.

Se puede acceder al objeto almacenada en la lista utilizando el índice que le correspondo.

```
words = ["Hola","mundo","!"]
print(words[0])
print(words[1])
print(words[2])
```

```
>>>
Hola
mundo
!
>>>
```

NOTA: El índice del primer objeto de la lista es cero (0)

Listas

¿Qué número imprime este código?.

```
nums = [5,4,3,2,1]
print(nums[1])
```

4

Listas

- Se puede crear una lista vacía, se hace con un par vacío de corchetes.

```
lista_vacia = []
print(lista_vacia)
```

```
>>>
[ ]
>>>
```

- Indexar fuera de los límites de los valores posibles de una lista genera un *IndexError.*

Listas

Normalmente una lista contiene objetos de un solo tipo, pero también es posible incluir varios tipos diferentes.

Las listas también pueden ser anidadas dentro de otras listas.

```
numero = 3
cosas = ["cadena",0,[1,2,numero],4.56]
print(cosas[0])
print(cosas[1])
print(cosas[2])
print(cosas[2][2])
print(cosas[3])
```

```
>>>
cadena
0
[1, 2, 3]
3
4.56
>>>
```

Listas

Algunos tipos, como las cadenas puede ser indexados como lista.

La indexación de **cadenas** se comporta como si estuviese indexando una lista que contiene los caracteres de la **cadena**.

```
var = "Hola mundo!"
print(var[6])
```

```
>>>
u
>>>
```

<u>NOTA</u>: Este tipo de indexaciones no es posible con otro tipo de dato

Listas

¿Cuál de las siguientes línea imprimirá error?

1. num = [5 , 4 , 3 , [2] , 1]
2. print (num[0])
3. print (num[3][0])
4. print (num[5])

- ○ Línea 3
- ○ Línea 2
- ● Línea 4

Operaciones de listas

Los objetos en un determinado índice de una lista pueden ser reasignados.

```
nums = [7,7,7,7,7]
nums[2] = 5
print(nums)
```

```
>>>
[7, 7, 5, 7, 7]
>>>
```

Operaciones de listas

Las listas pueden ser sumadas y multiplicadas de la misma manera que las cadenas.

```
nums = [1,2,3]
print(nums + [4,5,6])
print (nums * 3)
```

```
>>>
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>>
```

Operaciones de listas

Para verificar si un elemento está en una lista, se puede utilizar el operador **in**. Este devuelve **True** si el elemento está una o más veces en la lista y **False** si no.

```
words = ["hola","mundo","Bio","UdeA"]
print("hola" in words)
print("Bio" in words)
print("Ingenieria" in words)
print("udea" in words)
```

```
>>>
True
True
False
False
>>>
```

El operador in puede ser utilizado para determinar si una **cadena** es subcadena de otra

Listas

¿Cuál es el resultado de este código?.

```
nums = [10,9,8,7,6,5]
nums [0] = nums[1] – 5
if 4 in nums:
    print(nums[3])
else:
    print(nums[4])
```

7

Operaciones de listas

Para verificar si un objeto no está en una lista, se puede puede utilizar el operador **not** de alguna de las siguientes maneras

```
nums = [1,2,3]
print (not 4 in nums)
print (4 not in nums)
print (not 3 in nums)
print (3 not in nums)
```

```
>>>
True
True
False
False
>>>
```

Funciones y métodos de listas

Otra manera de alterar listas es utilizando el método **append**. Esto agrega un elemento al final de una lista ya existente.

```
nums = [1,2,3]
nums.append(4)
print(nums)
```

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

Listas

¿Cuál es el resultado de estas instrucciones?

```
word = [“hola”]
word.append(“mundo”)
print (word[1])
```

- mundo
- Error
- hola

Funciones y métodos de listas

Para obtener el número de elementos en una lista, se puede utilizar la función **len**.

```
nums = [1,3,5,2,4]
print(len(nums))
```

```
>>>
5
>>>
```

Listas

¿Cuál es el resultado de este código?.

```
letras = [“a”,“b”,“c”]
letras.append(“d”)
print(len(letras))
```

4

Funciones y métodos de listas

El método **insert** es parecido al de **append** excepto que te permite insertar un elemento en cualquier posición de una lista, al contrario de solo el final.

```
palabras = ["Python","Funciones"]
index = 1
palabras.insert(index,"Metodo")
print (palabras)
```

```
>>>
['Python', 'Metodo', 'Funciones']
>>>
```

Listas

¿Cuál es el resultado de este código?.

```
nums = [9,8,7,6,5]
nums.append(4)
nums.insert(2,11)
print(len(nums))
```

7

Funciones y métodos de listas

El método **index** encuentra la primera ocurrencia de un elemento de una lista y devuelve su índice.

Si el elemento no está en la lista, levanta una excepción ValueError.

```
letras = ['p','q','r','s','p','u']
print(letras.index('r'))
print(letras.index('p'))
print(letras.index('z'))
print(letras.index('Q'))
```

```
>>>
2
0
ValueError: 'z' is not in list
ValueError: 'Q' is not in list
>>>
```

Funciones de listas

Hay otras funciones y métodos que puede ser de utilidad.

max(lista): Devuelve el elemento de una lista con el máxima valor

min(lista): Devuelve el elemento de una lista con el mínimo valor

lista.**cont**(obj): Devuelve un conteo de cuántas veces un elemento está en una lista.

lista.**remove**(obj): Elimina un objeto de una lista

lista.**reverse**(): Invierte los elementos de una lista

Funciones y métodos de listas

```
letras = ['p','q','r','s','p','u']
numeros = [2,5,8,9,4,5,1,0,5]
print(max(numeros))
print(max(letras))
print(min(numeros))
print(min(letras))
print(numeros.count(5))
print(letras.count('p'))
print(numeros.remove(5))
#print(numeros.remove(3))
print(numeros)
print(letras.remove('r'))
print(letras)
print(numeros.reverse())
print(numeros)
print(letras.reverse())
print(letras)
```

```
>>>
9
u
0
p
3
2
None
ValueError: list.remove(x): x not in list
[2, 8, 9, 4, 5, 1, 0, 5]
None
['p', 'q', 's', 'p', 'u']
None
[5, 0, 1, 5, 4, 9, 8, 2]
None
['u', 'p', 's', 'q', 'p']
>>>
```

Funciones y métodos de listas

Rellene los espacios en blanco para agregar 'z' al final de una lista e imprimir la longitud de una lista.

lista.**appned**('z')

print(**len**(_lista_))

Rango

La función **range** crea una lista secuencial de número.

```
numeros = list (range(10))
print(numeros)

print(numeros[1])
```

```
>>>
[0,1,2,3,4,5,6,7,8,9]

1
>>>
```

Rango

Si **range** es llamado con un argumento. Produce un objeto con valores desde 0 hasta ese argumento.

Si se llama con dos argumentos, genera valores desde el primer argumento hasta el segundo.

```
numeros = list (range(3,8))
print(numeros)

print (range(20) == range(0,20))
```

```
>>>
[3,4,5,6,7]

True
>>>
```

Rango

¿Cuál es el resultado de estas instrucciones?

```
nums = list (range(5,8))
print (len(nums))
```

3

Rango

range puede tener un tercer argumento, que determina el incremento de la secuencia producida. Este tercer argumento debe ser un entero.

```
numeros = list (range(5,20,2))
print(numeros)
```

```
>>>
[5,7,9,11,13,15,17,19]
>>>
```

Listas

¿Cuál es el resultado de estas instrucciones?

```
nums = list(range(3,15,3))
print (nums[2])
```

- 9
- 0
- 3
- 12

Bucles y Listas

A veces se necesita ejecutar un código en cada elemento de la lista. Esto se llama iteración, y puede llevarse a cabo con un bucle **while** y una variable de contador.

```
palabras = ["Hola","Bioingenieros", "de la", "UdeA"]
cont = 0
max_index = len (palabras) - 1

while cont <= max_index:
    palabra = palabras[cont]
    print (palabra + "!")
    cont += 1
```

```
>>>
Hola!
Bioingenieros!
de la!
UdeA!
>>>
```

Bucle for

Iterar a través de una lista utilizando un bucle **while** requiere de bastante código, así que Python provee el bucle **for** como un atajo para lograr lo mismo.

```
palabras = ["Hola","Bioingenieros", "de la", "UdeA"]
for palabra in palabras:
    print (palabra + "!")
```

```
>>>
Hola!
Bioingenieros!
de la!
UdeA!
>>>
```

Bucle for

El bucle **for** es comúnmente utilizado para repetir algún código un un determinado número de veces. Esto se logra combinando los bucles **for** con objetos **range**.

```
for i in range(5):
    print ("Hola mundo!")
```

```
>>>
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
Hola mundo!
>>>
```

Bucle for

Rellenar los espacios en blanco para crear un bucle for que imprima solo los valores pares en el rango

```
for i in range ( 0 , 20 , 2 ):
    print ( i )
```

Calculadora

Crear una calculadora sencilla en Python, que permita escoger una de las cuatro operación aritméticas básicas e imprimir el resultado. En menú debe ser como se muestra a continuación.

1. Suma
2. Resta
3. Multiplicación
4. División
5. Salir

NOTA: Solo se puede salir si el usuario ingresa la opción **5. Salir**