3- Un postulante a un empleo, realiza un test de capacitación, se obtuvo la siguiente información: cantidad total de preguntas que se le realizaron y la cantidad de preguntas que contestó correctamente. Se pide confeccionar un programa que ingrese los dos datos por teclado e informe el nivel del mismo según el porcentaje de respuestas correctas que ha obtenido, y sabiendo que:

Nivel máximo: Porcentaje>=90%.
Nivel medio:Porcentaje>=75% y <90%.
Nivel regular: Porcentaje>=50% y <75%.
Fuera de nivel: Porcentaje<50%.

```
preg=int(input("Ingrese cantidad de Preguntas: "))
respu=int(input("Ingrese cantidadn respuestas correctas: "))
porcen=respu/preg*100
if porcen >=90:
    print("Nivel Máximo")
elif porcen >=75:
    print("Nivel Medio")
elif porcen >=50:
    print("Nivel Regular")
else:
    print("Fuera de Nivel")
```

4- Escribir un programa que solicite ingresar n notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores. Cuando ingresa nota=0 termina la ejecución

```
n=int(input("Ingrese cantidad de notas: "))
x=1
mayor=0
menor=0
while x <=n:
    nota=int(input("Ingrese nota: "))
    if nota==0:
        break;
    if nota>=7:
        mayor=mayor+1
    else:
        menor=menor+1
    x=x+1
print("Las notas mayores o iguales a 7 son: ", mayor)
print("Las notas menores a 7 son: ", menor)
```

5- Ingresan las alturas n personas y se calcula el promedio.
Preguntar cuántos datos se ingresarán

```
n=int(input("Ingrese cantidad de datos: "))
x=1
suma=0
while x <=n:
    alto=float(input("Ingrese áltura: "))
    suma=suma+alto
    x=x+1
print("El promedio de áltura es: ", suma/n)
```

6- En una empresa trabajan n empleados cuyos sueldos oscilan entre $100 y $500, realizar un programa que lea los sueldos que cobra cada empleado e informe cuántos empleados cobran entre $100 y $300 y cuántos cobran más de $300. Además el programa deberá informar el importe que gasta la empresa en sueldos al personal.

```
n=int(input("Ingrese cantidad de empleados: "))
x=1
suma=0
sueldo1=0
sueldo2=0
while x <=n:
    sueldo=int(input("Ingrese Sueldo: "))
    if sueldo<100 or sueldo>500:
        print("Error ingreso de datos")
        continue;
    elif sueldo > 300:
        sueldo2=sueldo2+1
    else:
        sueldo1=sueldo1+1
    suma=suma+sueldo
    x=x+1
print("Total gastado en sueldo: ", suma)
print("Sueldos mayores a $300: ", sueldo2)
print("Sueldos menores a $300: " ,sueldo1)
```

7- Se necesita realizar un control de edad de ingreso al Sistema de una empresa. Mientras la edad sea entre 18 y 65 pueden acceder al sistema caso contrario mostrar Mensaje de Acceso Denegado

```
acceso=int(input("Ingrese cantidad de ingresos: "))
x=1
while x<=acceso:
    edad=int(input("Ingrese edad: "))
    if edad < 18 or edad >65:
        print("ACCESO DENEGADO")
    else:
        print("Puede ingresar")
        x=x+1
```

¿Cuál es la diferencia?

x=x+1

```
acceso=int(input("Ingrese cantidad de ingresos: "))
x=1
while x<=acceso:
    edad=int(input("Ingrese edad: "))
    if edad < 18 or edad >65:
        print("ACCESO DENEGADO")
    else:
        print("Puede ingresar")
x=x+1
```

8- Una planta que fabrica perfiles de hierro posee un lote de n piezas. Confeccionar un programa que pida ingresar por teclado la cantidad de piezas a procesar y luego ingrese la longitud de cada perfil; sabiendo que la pieza cuya longitud esté comprendida en el rango de 1.20 y 1.30 son aptas. Imprimir por pantalla la cantidad de piezas aptas que hay en el lote.

```
canti=int(input("Ingrese cantidad de piezas: "))
x=1
aptas=0
while x <=canti:
    long=float(input("Ingrese longitud de la pieza: "))
    if long>=1.20 and long<=1.30:
       aptas=aptas+1
    x=x+1

print("Cantidad de piezas aptas: ", aptas)
```

Estructura repetitiva For

Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while, pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones que tenemos que recorrer un conjunto de datos.
En Python existen muchos objetos como por ejemplo las cadenas de texto (strings), las listas, los diccionarios, y las tuplas que son «iterables».
Esto significa que podemos iterar sobre sus elementos: cada uno de los caracteres de una cadena de texto, cada uno de los elemento de una lista, etc. Esta operación se realiza con un bloque de código llamado bucle *for* que indica las operaciones a realizar en cada iteración.

for *elemento* **in** *iterable*:

Funcionamiento

elemento es la variable que toma el valor del elemento dentro del *iterador* en cada paso del bucle. Este finaliza su ejecución cuando se recorren todos los elementos.
No es necesario definir la variable de control antes del bucle, aunque se puede utilizar como variable de control una variable ya definida en el programa.

Iterable puede ser una cadena de texto, listas, diccionarios, tuplas etc. También puede utilizarse la **función range()**

La sentencia *for* de Python itera sobre los ítems de cualquier secuencia (una lista o una cadenas de caracteres), en el orden que aparecen en la secuencia.
El cuerpo del bucle se ejecuta tantas veces como elementos tenga el **elemento recorrible** (elementos de una lista o de un **range()**, caracteres de una cadena, etc.)

Volvamos al ejemplo que vimos para while mostrar números del 1 al 100

```
x=1
while x<=100:
    print(x)
    x=x+1
```

```
for x in range(101):
    print(x)
```

La función *range()* acepta tres números enteros de los cuales dos son opcionales.
range(inicio, fin, paso)

Inicio: es el valor inicial de la secuencia (0 si no se especifica).

Fin: es el valor final de la secuencia, el cual no se incluye en el resultado. (condición de <)

Paso: indica el incremento entre elementos de la secuencia (1 si no se especifica).

Realizar un programa que muestre los números del 20 al 30.

```
for x in range(20,31):
    print(x)
```

La función range puede tener dos parámetros, el primero indica el valor inicial que tomará la variable x, cada vuelta del for la variable x toma el valor siguiente hasta llegar al valor indicado por el segundo parámetro de la función range menos uno.

Mostrar todos los números impares que hay entre 1 y 100.

```
for x in range(1,100,2):
    print(x)
```

La función range puede tener también tres parámetros, el primero indica el valor inicial que tomará la variable x, el segundo parámetro el valor final (que no se incluye) y el tercer parámetro indica cuanto se incrementa cada vuelta x.
En nuestro ejemplo la primer vuelta del for x recibe el valor 1, la segunda vuelta toma el valor 3 y así sucesivamente hasta el valor 99.

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio. Este problema ya lo desarrollamos, lo resolveremos empleando la estructura for para repetir la carga de los diez valores por teclado.

```
suma=0
for f in range(10):
    valor=int(input("Ingrese valor:"))
    suma=suma+valor
print("La suma es")
print(suma)
promedio=suma/10
print("El promedio es:")
print(promedio)
```

Como vemos la variable *f* del for solo sirve para iterar(repetir) las diez veces el bloque contenido en el for.
El resultado hubiese sido el mismo si llamamos a la funcion range con los valores: *range(1,11)*

Bucle For iterando elementos y cadenas de caracteres

```
for i in (1,2,3):
    print("Hola")
```

```
Hola
Hola
Hola
```

Ejecuta el print 3 veces pero cuidado 1,2, 3 son números enteros, NO es una variable contador.
El elemento iterable que tengo son tres números enteros entonces va a realizar tres(3) iteraciones.

Para probarlo vamos a ver el siguiente código:

```
for i in ("uno","dos","tres"):
        print("Hola")
```

```
Hola
Hola
Hola
```

Vemos que el resultado es el mismo porque también el iterable tiene tres(3) elementos.
O sea que la cantidad de veces a ejecutarse será también tres(3)

Ahora nos preguntamos ¿Qué valor toma la variable i? ¿Será el mismo?

Probamos el mismo código pero en lugar de `print("Hola")` ponemos `print(i)`

```
for i in (1,2,3):
    print(i)
```

```
====
1
2
3
```

```
for i in ("uno","dos","tres"):
    print(i)
```

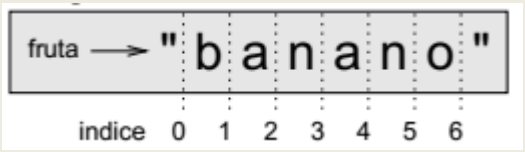
```
uno
dos
tres
```

¿Qué conclusiones sobre el funcionamiento del bucle **for** puedes sacar?

Tipo de Datos String (Cadenas de Caracteres)

Hasta aquí hemos visto tres tipos de datos: int, float y string.
Las cadenas (string) son cualitativamente diferentes de los otros dos tipos porque están compuestas de piezas más pequeñas, los caracteres. Son tipos de datos compuestos
Dependiendo de lo que hagamos podemos tratar un tipo compuesto como unidad o podemos acceder a sus partes. Esta ambigüedad es provechosa.

El operador corchete selecciona un carácter de una cadena.




```
fruta = "banano"
letra = fruta[1]
print(letra)
```

La expresión fruta[1] selecciona el carácter número 1 de fruta. La variable letra se refiere al resultado. Cuando desplegamos letra, obtenemos una pequeña sorpresa:

```
a
```

La primera letra de "banano" no es a. Se empieza a contar desde cero. La letra número 0 de "banano" es b. La letra 1 es a, y la letra 2 es n.

```
fruta = "banano"
letra = fruta[0]
print(letra)
```

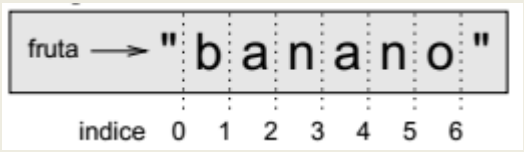
La expresión en corchetes se denomina índice. Un índice especifica un miembro de un conjunto ordenado, en este caso el conjunto de caracteres de la cadena. El índice puede ser cualquier expresión entera.

Longitud

La función *len* retorna el número de caracteres en una cadena

```
fruta="banano"
print(len(fruta))
```

El resultado será 6



Si pensamos que para acceder a la última posición escribimos el siguiente código

```
fruta="banano"
longitud=len(fruta)
ultima=fruta[longitud]
```

Al ejecutarlo nos dará error

```
    ultima=fruta[longitud]
IndexError: string index out of range
>>>
```

Como empezamos a contar desde cero, las seis letras se numeran de 0 a 5. En general, para obtener la ´ultima letra, tenemos que restar 1 a la longitud

```
fruta="banano"
longitud=len(fruta)
ultima=fruta[longitud-1]
```

Alternativamente, podemos usar índices negativos, que cuentan hacia atrás desde el fin de la cadena. La expresión **fruta[-1]** retorna la ultima letra **fruit[-2**] retorna la penúltima, y así sucesivamente.

Recorrido de cadenas

Muchos algoritmos implican procesar una cadena carácter por carácter. A menudo empiezan al inicio, seleccionan cada carácter en cada paso, le hacen algo y continúan hasta el final. Este patrón de procesamiento se denomina recorrido.
Hay una forma de realizarlo con la sentencia **while**:

```
fruta="manzana"
indice = 0
while indice < len(fruta):
    letra = fruta[indice]
    print (letra)
    indice = indice + 1
```

Este ciclo recorre la cadena y despliega cada letra en una línea independiente. La condición del ciclo es índice <len(fruta), así que cuando índice se hace igual a la longitud de la cadena, la condición es falsa, y el cuerpo del ciclo no se ejecuta. El ultimo carácter accedido es el que tiene el índice len(fruta)-1, es decir, el ultimo.

una sintaxis alternativa más simple el ciclo **for** :

```
fruta="manzana"
for caracter in fruta:
    print (caracter)
```

Segmentos de cadenas

Una porción de una cadena de caracteres se denomina **segmento**. Seleccionar un segmento es similar a seleccionar un carácter:

```
s="Pedro, Pablo y Maria"
print (s[0:5])

print(s[1:6])

print(s[7:12])

print(s[15:20])
```

```
==== RESTART:
Pedro
edro,
Pablo
Maria
>>>
```

El operador [n:m] retorna la parte de la cadena que va desde el carácter n hasta el m, incluyendo el primero y excluyendo el último

Si usted omite el primer índice (después de los puntos seguidos), el segmento comienza en el inicio de la cadena. Si se omite el segundo índice, el segmento va hasta el final.

File Edit Format Run

```
fruta="banana"
print (fruta[:3])

print(fruta[3:])
```

```
==== RESTAR
ban
ana
>>>
```

Comparación de cadenas

El operador de comparación funciona con cadenas. Para ver si dos cadenas son iguales

```
palabra=input("Ingrese fruta: ")
if palabra != "banana":
    print ("No hay bananas")

if palabra < "banana":
    print("Su palabra "+palabra+", va antes que banana")
elif palabra > "banana":
    print("Su palabra "+palabra+", va después que banana")

else:
    print("Su palabra es banana")
```

```
==== RESTART: C:/Users/Master/AppData/Loc
Ingrese fruta: pera
No hay bananas
Su palabra pera, va después que banana
>>>
```

Todas las letras mayúsculas vienen antes de las minúsculas. Si palabra vale ”Pera” la salida seria: Su palabra, Pera, va antes que banana.
Este problema se resuelve usualmente convirtiendo las cadenas a un formato común, todas en minúsculas por ejemplo, antes de hacer la comparación.

Métodos propios de las cadenas de caracteres

Los string tienen una serie de métodos (funciones aplicables solo a los string) que nos facilitan la creación de nuestros programas.
Los primeros tres métodos que veremos se llaman: **lower, upper** y **capitalize**.
upper() : devuelve una cadena de caracteres convertida todos sus caracteres a mayúsculas.
lower() : devuelve una cadena de caracteres convertida todos sus caracteres a minúsculas.
capitalize() : devuelve una cadena de caracteres convertida a mayúscula solo su primer caracter y todos los demás a minúsculas.

Las cadenas son inmutables

Uno puede caer en la trampa de usar el operador [] al lado izquierdo de una asignación con la intención de modificar un carácter en una cadena. Por ejemplo:

```
saludo="hola Mundo"
saludo[0]="H"
print (saludo)

saludo="Hola Mundo"
print(saludo)
```

Las cadenas son **inmutables**, lo que quiere decir que no se puede cambiar una cadena existente. Lo máximo que se puede hacer es crear otra cadena o asignar un nuevo valor.