Nivel 3 Introduccion a Programacion

Tomas Rodriguez - 202212868

Abril 2022

1 Intrucciones Iterativas - While

Las instrucciones iterativas son aquellas las cuales se necesitan repetir u determinado numero de veces o hasta que se cumpla una condicion.

1.1 While

```
While condicion:
    accion
    ...
    accion
    accion
```

Listing 1: Ciclo While

La instruccion **While** se determina por la condicion boolena presente en esta, es decir, mientras se cumpla la *condicion*, las acciones se van a ejecutar. Estas se dejaran de ejecutar una vez la condicion del **While** no se cumpla.
Una caracteristica fundamental de estos ciclos, es que dentro de si mismos se tiene que ir cambiando la variable la cual determina la condicion de ejecucion, de lo contrario, se formara un ciclo **infinito**.

```
i  = 0
while i < 10:
    print(i)
print("Fin")
```

Listing 2: While infinito

como se observa en el anterior codigo, la vaiable **i** jamas va a cambiar, por ende siempre sera menor a 10 lo que generara un ciclo infinito. La forma correcta de modelar este codigo es cambiando la variable que determina la condicion de la siguente manera:

```
i  = 0
while i < 10:
    print(i)
    i += 1
print("Fin")
```

Listing 3: While Finito

Con este pequeño cambio, la variable podra llegar a 10 y el bucle llegara un momento en el que no se ejecute mas.

1.1.1 Los Centinelas

Los centinelas, son variables las cuales nos permiten controlar las ejecuciones de un ciclo. Normalmente es mas complejo pensar en condiciones de parada de un ciclo que las de continuacion, entonces, los centinelas los cuales normalmente son *Booleanos*, nos permiten simplificar esto:

```
def mcd(n1,n2):
n = min(n1,n2)
encontrado = False
while encontrado == False:
    if (n1%n == 0 and n2%n == 0):
        encontrado = True
    else:
        n -= 1
return n
```

Listing 4: While Finito

Lo que hace el codigo anterior es encontrar el MCD entre dos numeros, se sabe que el MCD de dos numeros es aquel que cuando se dividen ambos numeros entre otro y el residuo de ambos es 0, entonces ese es el MCD, por ende, es mas facil pensar en esa condicion que utilizar leyes de morgan (Nivel 2) para cambiar la expresion booleana.

2 Cadenas de caracteres o Strings 2

Aparte de lo visto en el modulo 2 del curso, hay algunas caracteristicas adicionales sobre los **strings** las cuales se deben conocer.

2.1 Indexaciones

Esto consiste en la capacidad de acceder a un elemento especifico de una cadena, ejemplo:

```
string = "Hola Mundo"
print(string[1])
```

Listing 5: Indexacion String

El resultado del codigo anterior va a ser "o", ya que en programacion, todos los elementos que son iterables, es decir que se pueden recorrer con ciclos, inician desde el indice 0, por esto mismo, el ultimo elemento de la cadena no es:

```
string[len(string)]
```

Listing 6: Indexacion String 2

si no sera:

```
string[len(string)-1]
```

Listing 7: Indexacion String 3

Si se interara usar la forma de acceder sin el -1 entonces python hara raise de un error llamado **IndexError**.
Otra forma de buscar caracteres en un string es con el metodo **find**. Este metodo recibe una cadena y nos indica la posicion en la que esta se encuentra, sin embargo, si no encuentra la cadena, devuelve -1.

```
string = 'Hola'
string.find('H')
```

Listing 8: Metodo Find

2.2 Sub-Cadenas

Las subcadenas son partes de una cadena principal, es decir son recortes o partes mas pequeñas de la cadena original. Para obtener subcadenas se hacer por medio del operador de **slicing** el cual consiste en cortar un string.

```
x = 'Piratas'
s = x[n:m]
```

Listing 9: Slicing String 1

n y m son numeros que indican n el indice de inicio y hasta donde se recorta m-1 caracteres. Si se omite alguno de los numeros de inicio o fin es decir que la subcadena va desde el inicio o va hasta el final.

```
x = 'Piratas'
s = x[n:]
z = x[:m]
```

Listing 10: Slicing String 2

Ahora bien, existe un tipo de **slicing** mas avanzado y es implementando pasos. Estos slices se ven de la siguente manera:

```
x = 'Piratas'
s = x[n:m:a]
```

Listing 11: Slicing String 3

En este se usa un nuevo valor **a** el cual indica que tantos caracteres se tienen que omitir, se ilustra mejor a continuacion:

```
x = 'Piratas'
s = x[::2]
print(s)
```

Listing 12: Slicing String 4

El resultado de este codigo sera: **Prts**
Ahora bien, cabe resaltar que las cadenas son **inmutables**, es decir, no se pueden cambiar los valores de una posicion ya definida, para ello se tiene que usar slicing para crear 2 subcadenas y poner el nuevo caracter en medio:

```
x = 'Hola mi jente'
correccion = x[:8] + 'g' + x[9:]
```

Listing 13: Inmutabilidad

El resultado de este codigo sera: Hola mi gente

3 Instrucciones Iterativas - For

Asi como la instruccion While, el **For** es una instruccion iterativa, se podria decir que este tiene 2 tipos distintos

- in range: Funciona como el While con los indices de los elementos
- in: funciona recorriendo ya no los indices sino aquellos elementos que componen uno mas grande.

```
x = 'Hola mi jente'
for i in range(len(x)):
    print(x[i])
```

Listing 14: For in range

```
x = 'Hola mi jente'
for caracter in x:
    print(caracter)
```

Listing 15: For each

Estos dos son equivalentes, solo que uno es con indices y el otro con los sub-elementos que conforman a uno mas grande. Ahora bien, en el caso del **For in range**, la funcion **range()** puede recibir 3 parametros al igual que los slices:

- Primer Parametro: Indice de inicio
- Segundo Parametro: Indice de Fin
- Tercer Parametro: Saltos de indices

4 Listas

Las listas son colecciones de valores y en python es una estructura de datos. Aquellos elementos que conforman una lista son llamados elementos o items, cabe resaltar que en **python** las listas pueden albergar cualquier tipo de dato. Las listas son muy similiares a los string, ya que al fin al cabo son secuencias de elementos o caracteres (caso String), pero, ademas estos comparten muchas operaciones ya conocidas:

1. Pueden ser parametros de funciones
2. Se puede usar la funcion *len()* para conocer la cantidad de elementos en la lista
3. El operador de suma + se puede usar para concatenar dos listas distintas
4. Se pueden indexar los elementos de la lista
5. Se puede usar el operador de corte o slicing
6. Se puede usar las instrucciones iterativas sobre estas
7. Funciones *max() y min()* funcionan en listas

Ahora bien, las listas tienen diversas formas de iniciarse o declararse, algunas de estas son:

```
a = []
```

Listing 16: Declaracion de una lista

```
a = list(range(1,4))
```

Listing 17: Combinacion de List y Range para generar una lista

Este codigo va a generar el siguente array:

[1 2 3]

```
a = [0]*10
```

Listing 18: Operador de multiplicacion para generar una lista

Este codigo va a generar el siguente array:

[0 0 0 0 0 0 0 0 0 0]

4.1 Operaciones de Listas

Asi como en las cadenas, las listas tienen operaciones muy parecidas a aquellas que se realizan sobre las cadenas de caracteres, algunas de estas son:

```
if lista1 == lista2:
```

Listing 19: Comparacion de Listas

La comparacion del operador == en listas funciona de la siguente forma:

- Primero compara el tamaño de cada lista, si este es diferente devuelve False.
- Si pasa el primer filtro, despues compara elemento por elemento de izquierda a derecha, si un solo elemento es diferente devuelve False.

Ahora bien, estas dos reglas se cumplen para todas las operaciones de comparacion como $<$, $<=$, $>$, $>=$ con sus repesectivas definiciones claro esta solo que en orden inverso, primero se revisan los elementos y despues el tamaño de la lista.

```
lista1[0] = 1
lista1[1:3] = [2,3]
```

Listing 20: Modificaciones en Listas

Para modificar elementos que estan dentro de una lista se usa la operacion de **indexacion** con el fin de alterar o modificar el valor que se encuentra en dicha posicion o incluso se puede usar la operacion de **slicing** para llevar a cabo estas modificaciones.

```
del lista1[0]
del lista1[1:3]
```

Listing 21: Eliminacion de elementos de Listas

Ahora bien, para las operaciones de eliminacion se usa la palabra reservada de **del** la cual indica delete, esta se puede combinar con las operaciones de modificacion de listas antes vistas.

```
a = [1,2,3]
b = a[:]
```

Listing 22: Clonacion de Listas

Esta operacion se conoce como clonacion de listas, es decir, se va a hacer una copia de la lista de la variable a en la variable b.

4.1.1 Operador is

```
a = [1,2,3]
b = [1,2,3]
c = a
```

Listing 23: Eliminacion de elementos de Listas

En este codigo si se hace un codigo de evaluacion para ver si las listas a, b, c son iguales, el resultado va a ser $True$ ya que todas las variables tienen como valor $[1, 2, 3]$, sin embargo, si usamos este nuevo operador is tenemos como resultado lo siguente:

```
a is b
b is c
a is c
```

Listing 24: Operador is en Listas

El resultado de evaluar dichas condiciones es:

1. False
2. False
3. True

Esto debido a que si nos vamos a observar que pasa en la memoria de nuestro computador en el *Listing* 22 estara pasando lo siguente:

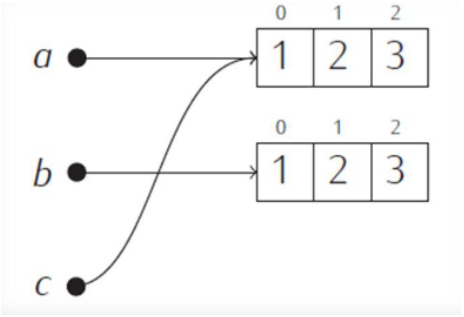

Figure 1: Representacion Memoria

Esto nos indica que, tecnicamente las variables a y c tienen un unico valor compartido, sin embargo, b tiene un unico valor para si mismo, por ello, al usar la operacion is tan solo la expresion $a\ is\ c$ da como resultado $True$.
Ahora bien, en el caso de la operacion de **clonacion**, la lista clonada en la otra variable no va a estar referenciada al mismo lugar en memoria si no que cada lista tendra su lugar en memoria distinto.

4.2 Metodos de Listas

Asi como las listas poseen operaciones, estas tambien poseen unos metodos los cuales se escriben de la siguente manera:

```
1 lista.metodo(parametros)
```

Listing 25: Metodos de Listas

Los metodos son funciones que hacen parte de las listas, es decir, estas tambien pueden tener parametros, retornos, etc. algunos de los mas importantes son:

- append(elemento): Adiciona un elemento al final de la lista
- insert(i,elemento): Adiciona un elemento en la posicion i de la lista
- count(elemento): Cuenta cuantas veces se repite un elemento en la lista
- extend(listaNueva): Junta dos listas, aquella a la cual se le aplica el metodo y la que se pasa por parametro, para que aquella que es parametro se agregren sus elementos al final de la lista que invoca el metodo

- index(elemento): Devuelve el indice de la primera ocurrencia en donde se encuentra el elemento.
- reverse(): Le da la vuelta a todos los elementos, es decir, invierte el orden de la lista
- sort(): ordena los elementos en la lista
- remove(elemento): elimina la primera ocurrencia del elemento en la lista
- copy(): Hace una copia de la lista
- pop(indice): Dado un indice, elimina el elemento que pertenece a dicho indice y lo devuelve
- clear(): Elimina todos los elementos de una lista

4.3 De cadenas a listas y viceversa

Ahora bien, despues de conocer las operaciones y metodos de las listas, hay un uso adicional que estas tienen y es la capacidad de separar y hacer strings a partir de sus elementos.

```
1 cadena = "uno dos tres"
2 lista = cadena.split(" ")
```

Listing 26: Metodo Split de cadena a lista

En este codigo se usa el metodo *split* de las cadenas de caracteres para separar la cadena en una lista que da como resultado: $[uno, dos, tres]$, esto fue posible gracias a que en la cadena, las palabras estaban separadas por espacion en blanco, por ende, el parametro que usa el metodo *split* es el separador de la cadena, es decir, porque estan separadas las palabras o elementos.

```
1 cadenaReconstruida = " ".join(lista)
```

Listing 27: Metodo Join de Lista a cadena

Ahora bien, el metodo inverso a *split* es el metodo *join*, el cual como indica su nombre es juntar los elementos de la lista en un unico string.

5 Recorridos en listas y diccionarios

Los recorridos para estas dos estructuras de datos son similares pero a la vez diferentes por la naturaleza de cada uno.

5.1 Recorridos en listas

5.1.1 Recorridos Totales

Este tipo de recorridos se llevan a cabo con instrucciones **While**, **For in range** y **For each**. Estos recorridos son aquellos en los cuales se va recorriendo los indices o elementos de una lista uno por uno hasta el final de la lista.

5.1.2 Recorridos Parciales

Este tipo de recorridos son lo contrario a los totales, en estos, se busca que la cantidad a elementos a recorrer sea lo minimo hasta cumplir cierto objetivo, es decir, su funcion es que se cumpla una condicion con el fin de darles parada y que no se lleven hasta el final de la lista. Los unicos recorridos capaces de lograr esto son el **While** y el **While con centinela**, algunos ejemplos son:

```
def posicion_recorrido_parcial_con_centinela(lista: list, buscado:
    int) -> int:
    i = 0
    posicion = -1
    ya_encontre = False
    while i < len(lista) and ya_encontre == False:
        if lista[i] == buscado:
            ya_encontre = True
            posicion = i
        i += 1
    return posicion
```

Listing 28: While con centinela

```
def posicion_recorrido_parcial_sin_centinela(lista: list, buscado:
    int) -> int:
    i = 0
    while i < len(lista) and lista[i] != buscado:
        i += 1
    if i == len(lista):
        posicion = -1
    else:
        posicion = i
    return posicion
```

Listing 29: While sin centinela

5.2 Recorridos en diccionarios

En el caso de los diccionarios existen tambien 2 tipos de recorridos, aquellos que son llave y aquellos que son por valor, sin embargo, estos recorridos usan los mismos tipos de recorridos que las listas, ya que para recorrer los diccionarios por llaves se usa el metodo *keys*() para que se recorra una lista de las llaves del diccionario, o si es por valor, se usa el metodo *values*() que tambien devuelve una lista con los valores para ser recorridos.

6 Manejo de archivos

Los archivos son objetos donde podemos llevar a cabo operaciones de lectura y escritura de informacion. En programacion, para usar dichos objetos se deben llevar a cabo 3 pasos:

1. Abrir el archivo: funcion *open*(*nombreArchivo*, *accion*), donde la accion puede ser: "r" de read, "w" de write o "a" de adicion

2. Leer o escribir la informacion: Dependiendo la accion en la funcion *open* se usa *readline* o *write*

3. Cerrar el archivo: Se realiza con la funcion *close*