Ampliando conocimientos de Python

Este cuaderno forma parte del curso de Iniciación a la programación con Python del programa de Open Course Ware (OCW) de la Universidad de La Laguna.

Condicionales

Las estructuras condicionales permiten tomar decisiones en función de ciertas condiciones. Se utilizan para ejecutar diferentes bloques de código dependiendo de si una condición dada es verdadera o falsa.

En Python se utiliza la palabra `if` para comenzar el condicional, `elif` para establecer una condición si las condiciones anteriores no son `True` y `else` si ninguna de las condiciones anteriores se cumplen. Por ejemplo:

```
x=4
y=9
if x>y:
  print("x es mayor que y") #Se ejecuta cuando el valor de x (4) es
mayor que y (9).
elif x<9:
  print("x es menor que 9") #Se ejecuta cuando el valor de x (4) sea
menor que 9.
else:
  print("x no es mayor que y ni menor que 9") #Se ejecuta cuando
ninguna de las condiciones anteriores se cumpla.
```

Puedes también colocar un condicional dentro de otro condicional:

```
x=7
y=2
if x>y:
  print("x es mayor que y") #Se ejecuta cuando el valor de x (4) es
mayor que y (9).
  if x>5:
    print("x es mayor que 5") #Se ejecuta cuando se cumpla la
condición anterior y si el valor de x (7) es mayor que 5.
elif x<9:
  print("x es menor que 9") #Se ejecuta cuando el valor de x (4) sea
menor que 9.
else:
  print("x no es mayor que y ni menor que 9") #Se ejecuta cuando
ninguna de las condiciones anteriores se cumpla.
```

Puedes probar este tema en la siguiente celda de código.

```
a=7.9
b=2
c=6
if a>b<c:
  print(a,"es mayor que",b,"que es menor que",c) #Se ejecuta cuando a
es mayor que b y c mayor que b.
elif a<b:
  print(a,"es menor que",b) #Se ejecuta cuando no se cumple la primera
condición y b es mayor que a.
elif b>c:
  print(b,"es mayor que",c) #Se ejecuta cuando no se cumple ni la
primera condición ni la segunda y b es mayor que c.
else:
  print("Los valores introducidos no cumplen ninguna condición") #Se
ejecuta cuando no se cumple ninguna de las condiciones, por ejemplo,
que los tres números sean iguales.
```

Listas

Las listas permiten almacenar varios datos en una única variable. Los elementos se almacenan ordenados, pueden cambiar los valores almacenados y permiten valores duplicados.

Las listas se definen utilizando corchetes `[]`. Por ejemplo:

```
x=[1,3.4,5]
print(x) #Sale por pantalla [1, 3.4, 5].
```

Es posible diferentes tipos de datos en la misma lista.

Para acceder a un elemento concreto, se utilizan también corchetes. Es necesario indicar la posición con un número entero. Los índices empiezan en `0`. Por ejemplo:

```
x=[12,"hola",4.5,9]
print(x[1]) #Sale por pantalla 'hola'.
print(x[-1]) #Sale por pantalla el último elemento, 9 en este caso.
print(x[1:3]) #Sale por pantalla el elemento con índice 1 (inclusivo),
y el elemento con índice 3 (exclusivo), ['hola', 4.5] en este caso.
print(x[:3]) #Sale por pantalla desde el primer elemento (índice 0),
hasta el segundo, [12, 'hola', 4.5] en este caso.
```

Con las listas es posible realizar una serie de funciones como:

- `append()`: agregar elemento al final de la lista.
- `insert()`: insertar un elemento en una posición concreta.
- `remove()`: eliminar el primer elemento cuyo valor coincida.
- `sort()`: ordena una lista. Por defecto, este orden es alfanumérico de manera ascendente.
- `reverse()`: revierte el orden de los elementos dentro de la lista.

- `len()`: devuelve la cantidad de elemento almacenados en una lista.
- `max()`: devuelve el elemento con el valor más alto.
- `min()`: devuelve el elemento con el valor más bajo.
- `sum()`: devuelve la suma de todos los elementos de la lista.

Un ejemplo de estas funciones:

```
mi_lista = [] #Lista vacía.

mi_lista.append(5)
mi_lista.append(2)
mi_lista.append(8)
print(mi_lista) #Sale por pantalla [5, 2, 8].

mi_lista.insert(1, 10)
print(mi_lista) #Sale por pantalla [5, 10, 2, 8].

mi_lista.remove(2)
print(mi_lista) #Sale por pantalla [5, 10, 8].

mi_lista.sort()
print(mi_lista) #Sale por pantalla [5, 8, 10].

mi_lista.reverse()
print(mi_lista) #Sale por pantalla [10, 8, 5].

print(len(mi_lista)) #Sale por pantalla 3.
print(max(mi_lista)) #Sale por pantalla 10.
print(min(mi_lista)) #Sale por pantalla 5.
print(sum(mi_lista)) #Sale por pantalla 23.
```

Es posible conocer si un elemento se encuentra dentro de una lista utilizando el operador `in`. Por ejemplo:

```
nombres = ["Diego", "Antonio", "Pilar", "Lucía"]
print("Diego" in nombres) #Devuelve True.
print("Celia" in nombres) #Devuelve False.
```

Puedes probar este tema en la siguiente celda de código.

```
mi_lista = [3, 3, "Python"] #Lista nueva con valores de diferentes
tipos.

mi_lista.append(7)
mi_lista.append("Hola") #Añadir 7 y 'Hola' al final de la lista.

mi_lista.insert(2, 2.0) #Añadir 2.0 en el tercer elemento.

mi_lista.remove(3) #Eliminar el primer 3 que aparece.
```

```
mi_lista.reverse() #Revertir la lista.

longitud = len(mi_lista) #Obtener la cantidad de elementos de la
lista.

primer_elemento = mi_lista[0] #Obtener el primer elemento de la lista.
tercer_elemento = mi_lista[2] #Obtener el tercer elemento de la lista.

primeros_dos = mi_lista[:2] #Obtener los dos primeros elementos de la
lista.
ultimos_dos = mi_lista[-2:] #Obtener los dos últimos elementos de la
lista.

print("Lista resultante:", mi_lista)
print("Longitud de la lista:", longitud)
print("Primer elemento:", primer_elemento)
print("Tercer elemento:", tercer_elemento)
print("Primeros dos elementos:", primeros_dos)
print("Últimos dos elementos:", ultimos_dos)

numeros = [3, 7, 2]

print("El elemento más grande de la lista es:", max(numeros))
print("El elemento más pequeño de la lista es:", min(numeros))
print("La suma de los elementos de la lista es:", sum(numeros))
print("¿Está el número 7 en la lista?", 7 in numeros)
print("¿Está el número 21.3 en la lista?", 21.3 in numeros)
```

Rangos

Son secuencias inmutables de números que se utilizan habitualmente en los bucles y para generar secuencias numéricas. Se utiliza la palabra `range()` para trabajar con ellos. Por ejemplo:

```
x=range(6) #Crea un rango del 0 al 5 (0,1,2,3,4,5).
x=range(1,6) #Crea un rango del 1 al 5 (1,2,3,4,5).
x=range(1,9,2) #Crea un rango del 1 al 7, de 2 en 2 (1,3,5,7).
```

Puedes probar este tema en la siguiente celda de código.

```
x=range(4,80,5)
for i in x:
  print(i) #Sale por pantalla desde el 4 al 80 (no inclusivo) de 5 en
5, `(4,9,14,24,29,34,39,44,49,54,59,64,69,74,79)`.
```

Bucles

Son estructuras de control que permiten ejecutar un bloque de código repetidamente, mientras se cumpla una condición o en función de una secuencia que le pasemos al bucle. En la celta de código anterior, se usó un bucle para imprimir por pantalla cada uno de los elementos del rango generado.

Existen dos tipos de bucles en Python.

Bucles for

Sirven para iterar sobre una secuencia, como son las cadena de texto, los rangos, las listas o cualquier objeto que sea iterable. El bucle `for` recorre cada elemento de la secuencia y ejecuta un bloque de código para cada elemento. Por ejemplo:

```
x=["Yaiza", "Ayoze", "Gara"]

for i in x:
    print("¡Hola,", i, "!") #Sale por pantalla `¡Hola, Yaiza !`;
`¡Hola, Ayoze !` y `¡Hola, Gara !`.
```

En el bucle anterior, la variable `i` contiene en la primera iteración el primer elemento de la lista `x`, es decir, "Yaiza". En la segunda iteración la `i` valdrá "Ayoze". En la tercera iteración la `i` valdrá "Gara". Al llegar al final de la lista, terminará la ejecución del bucle.

Bucles while

Este tipo de bucle se ejecuta mientras una condición dada sea verdadera. El bucle `while` verifica la condición tras cada iteración y termina la ejecución si la condición es falsa. Por ejemplo:

```
cnt=0
while cnt<3:
  print("Este bucle se ha ejecutado",cnt+1,"vez/veces.") #Sale por
pantalla 'Este bucle se ha ejecutado 1 vez/veces.'; 'Este bucle se ha
ejecutado 2 vez/veces.' y 'Este bucle se ha ejecutado 3 vez/veces.'.
  cnt+=1
```

Bucles anidados

Al igual que los condicionales, es posible ejecutar un bucle dentro de otro (anidados). Por ejemplo:

```
x=["Yaiza", "Ayoze"]
y=["Arucas", "Curbelo", "Chinea"]
for nombre in x:
  for apellido in y:
    print("¡Hola,", nombre, apellido, "!") #Sale por pantalla `¡Hola,
Yaiza Arucas !`; `¡Hola, Yaiza Curbelo !`; `¡Hola, Yaiza Chinea !`;
```

```
`¡Hola, Ayoze Arucas !`; `¡Hola, Ayoze Curbelo !` y `¡Hola, Ayoze Chinea !`.
```

Else, break y continue

La palabra `else` permite ejecutar un fragmento de código cuando termina la ejecución del bucle, bien porque haya terminado de iterar o bien porque la condición ya no se cumpla. Por ejemplo:

```
n=range(2,11,2) #[2, 4, 6, 8, 10]
suma=0
for i in n:
    suma+=n
else:
    print("La suma de los números pares es:", suma) #Sale por pantalla la suma de 2+4+6+8+10=30, `La suma de los números pares es: 30` cuando finaliza la ejecución del bucle.

cnt = 0
while cnt < 5:
    print("Contador:", cnt) #Sale por pantalla la iteración del bucle como `Contador: 0`; `Contador: 1`; `Contador: 2`; `Contador: 3` y `Contador: 4`.
    cnt += 1
else:
    print("El contador ha alcanzado", cnt, "y el bucle while ha terminado") #Sale por pantalla, cuando finaliza el bucle, `El contador ha alcanzado 5 y el bucle while ha terminado`.
```

La palabra `break` permite terminar la ejecución de un bucle sin que este haya finalizado o aunque se siga cumpliendo la condición de parada. Por ejemplo:

```
frutas = ["manzana", "banana", "cereza", "uva"]
for i in frutas:
    if i == "cereza":
        break #Termina el bucle si aparece la palabra cereza.
    print(i) #Sale por pantalla `manzana` y `banana`.

n = 1
while n <= 5:
    if n == 3:
        break #Termina el bucle si aparece el número 3.
    print(n) #Sale por pantalla `1` y `2`.
    n += 1
```

Si se ejecuta un `break` dentro de un bucle, y este tiene un `else`, este último fragmento de código no se llegará a ejecutar. Por ejemplo:

```
frutas = ["manzana", "banana", "cereza", "uva"]
for i in frutas:
    if i == "cereza":
        break #Termina el bucle si aparece la palabra cereza.
    print(i) #Sale por pantalla `manzana` y `banana`.
else:
  print("He terminado de imprimir la lista de frutas.") #Este texto
nunca saldrá por pantalla si cereza está en la lista.
```

La palabra `continue` permite saltar la iteración actual en la que se encuentra un bucle, pasando a la siguiente y sin ejecutar el código que se encuentre bajo la palabra. Por ejemplo:

```
frutas = ["manzana", "banana", "cereza", "uva"]
for i in frutas:
    if i == "cereza":
        continue #Salta a la siguiente iteración si aparece la palabra
cereza.
    print(i) #Sale por pantalla `manzana`, `banana` y `uva`.

n = 1
while n <= 5:
    if n == 3:
        continue #Salta a la siguiente iteración del bucle si aparece
el número 3.
    print(n) #Sale por pantalla `1`, `2`, `4` y `5`.
    n += 1
```

Puedes probar este tema en la siguiente celda de código.

```
for i in range(1, 6):
    if i == 3:
        continue  #Salta la iteración actual cuando i sea igual a 3.
    elif i == 5:
        break  #Sale del bucle cuando i sea igual a 5.
    print("Bucle for - Iteración:", i)

j = 0
while j < 5:
    j += 1
    if j == 2:
        continue  #Salta la iteración actual cuando j sea igual a 2.
    elif j == 4:
        break  #Sale del bucle cuando j sea igual a 4.
    print("Bucle while - Iteración:", j)
```

Funciones

Una función se define como una porción de código reutilizable dentro de un programa. En Python se definen utilizando la palabra `def`, seguido de paréntesis. Las funciones pueden tomar

entradas, denominadas argumentos, y devolver algún dato, valor de retorno, aunque no siempre es necesario. Por ejemplo:

```
def saludar():
    return "¡Hola, Mundo!" #Definición de la función saludar que no recibe argumentos y devuelve el texto `¡Hola, Mundo!`.

print(saludar()) #Llamada a la función saludar para que se ejecute. Sale por pantalla `¡Nola, Mundo!`.
```

A la función es posible pasarle argumentos para poder ejecutar instrucciones con esos datos. Estos se ponen entre los paréntesis y se trabaja con ellos como si fuesen variables. Por ejemplo:

```
def saludar(nombre):
  return "¡Hola, "+nombre+"!" #Devuelve el texto `¡Hola,` junto con el texto que ponga el usuario en el argumento, seguido de `!`.

print(saludar("Jorge")) #Llamada a la función saludar con el argumento `Jorge`. Sale por pantalla el texto `¡Hola, Jorge!`.
```

También es posible pasar más de un argumento, para lo que se deben separar cada uno por comas. Por ejemplo:

```
def sumar(a, b, c):
    return a+b+c #Devuelve la suma de 3 números o concatena 3 cadenas de texto.

print("La suma es:", sumar(3, 5.8, 7)) #Llamada a la función sumar. Sale por pantalla 15.8.
```

Es posible establecer valores por defecto para los argumentos. Por ejemplo:

```
def saludar(nombre="Desconocido"):
  return "¡Hola, "+nombre+"!" #Devuelve el texto `¡Hola,` junto con el texto que ponga el usuario en el argumento, seguido de `!`. En caso de que no llame a la función con un argumento, devolverá la cadena `¡Hola, Desconocido!`.

print(saludar("Jorge")) #Llamada a la función saludar con el argumento `Jorge`. Sale por pantalla el texto `¡Hola, Jorge!`.
print(saludar()) #Llamada a la función saludar sin argumentos. Sale por pantalla el texto `¡Hola, Desconocido!`.
```

Puedes probar este tema en la siguiente celda de código.

```
def area(base, altura):
    return base*altura #Devuelve el área de un rectángulo.

def par(n):
```

```
    return n%2==0 #Devuelve si el número es par o no.

print("Área del rectángulo:", area(5,8)) #Sale por pantalla `Área del rectángulo: 40`.
print("Área del cuadrado:", area(5,5)) #Sale por pantalla `Área del cuadrado: 25`.

n = 7
if par(n):
    print(n, "es un número par.") #Si n (7) fuese par, saldría por pantalla `7 es un número par`.
else:
    print(n, "no es un número par.") #Sale por pantalla `7 no es un número par`.
```