Parte I - Sesión 2

June 29, 2025

1 Python Científico | Parte I / Sesión 2

Alan Badillo Salas

Ciudad de México, 28 de junio de 2025

1.1 Contenido

1. Valores, Variables y Tipos de datos
2. Expresiones Lógicas y Artiméticas
3. Sentencias, Bloques y Estructuras de Control
4. Colecciones: Listas, Tuplas y Diccionarios
5. Funciones: Sin parámetros y con parámetros

1.2 Condicionales

Las condicionales son estructuras de control que permiten ejecutar un **bloque de código** cuando una condición es verdadera.

La estructura permite anidar condiciones alternativas en caso de que la condición sea falsa (decisión en cascada).

Sintaxis de una condicional simple

```
if <condición>:
  <sentencias>
```

Donde las sentencias que queremos ejecutar dependerán de sí la condición es verdadera (la condición es un valor lógico) y deberán estar anidadas dentro del bloque marcado por los dos puntos (`:`) a esto se le conoce como nivel de indentación.

Un nivel de indentación es reconocido como las sentencias espaciadas uniformemente dentro del bloque bajo la misma alineación.

[ ]:
```
x = int(input("Dame x: "))

if x >= 0:
  print(f"Tu x={x} es positiva")
```

```
Dame x: -12
```

La estructura condicional `if` puede anidar una estructura dual llamada `else` que ejecutará la contraparte, es decir, ejecutará un bloque cuando no se alcance la verdad (un valor lógico verdadero).

Sintaxis de una condicional doble (dual)

```
if <condición>:
  <sentencias verdaderas>
else:
  <sentencias opuestas>
```

[ ]:
```python
x = int(input("Dame x: "))

if x >= 0:
  print(f"Tu x={x} es positiva")
else:
  print(f"Tu x={x} es negativa")
```

```
Dame x: -12
Tu x=-12 es negativa
```

1.2.1 Ejemplo: Determinar las raices de una ecuación cuadrática

La ecuación cuadrática $ax^2 + bx + c$ tiene dos posibles soliciones alcanzadas tras completar el binomio cuadrado y despejar a x:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Generando la solución positiva y negativa, en caso de que el discriminante $b^2 - 4ac \geq 0$, es decir, solo con un discriminante positivo existirá una solución en los reales $\mathbb{R}$.

[ ]:
```python
# Inicio

a = float(input("a: "))
b = float(input("b: "))
c = float(input("c: "))

e = None
x1 = None
x2 = None

if a == 0:
  e = "División entre cero (`a` es cero)"
else:
  d = b ** 2 - 4 * a * c
  if d >= 0:
    x1 = (-b + d ** 0.5) / (2 * a)
    x2 = (-b - d ** 0.5) / (2 * a)
  else:
    e = "Discriminante negativo (solición imaginaria)"

# Fin
```

```
e, x1, x2
```

1.3 Estructura condicional cíclica

Cuando requerimos repetir un **bloque de sentencias** determinadas por una condición podemos usar la estructura de control condicional cíclica, esta nos permitirá evaluar una condición y si se cumple ejecutará las sentencias anidadas y luego volverá a comprobar la condición y si es verdadera volverá a ejecutar la senticias, repitiendo cíclicamente el procedimiento hasta que de despúes de ejecutar el bloque y comprobar la condición esta ya no se cumpla.

Es decir, mantendremos ejecutando un bloque de sentencias *mientras* la condición se cumpla.

Sintaxis de una condicional cíclica

```
while <codición>:
  <sentencias>
```

[ ]:
```
n = 1

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

```
1
2
3
4
```

[ ]:
```
x = 1

while x < 10000:
  print(x)
  x = 3 * x - 1
```

```
1
2
5
14
41
122
365
1094
3281
9842
```

1.4 Listas

Las listas son colecciones indexadas que pueden almacenar cero o más elementos y permiten seguir agregando, reemplazando y quitando elementos.

A través del índice podemos acceder a los elementos almacenados, o también podemos usar una serie de funciones que se aplican directamente a la listas ya que las listas son secuencias de elementos:

Funciones secuenciales

- `min(<secuencia>)` - Recupera el valor mínimo de la secuencia (lista)
- `max(<secuencia>)` - Recuepera el valor máximo de la secuencia (lista)
- `len(<secuencia>)` - Recupera el tamaño o número de elementos de la secuencia (lista)
- `sum(<secuencia>)` - Calcula la suma de todos los elementos en la secuencia (lista)

Además para manipular la lista podemos usar algunos operadores para agregar o quitar elementos:

Operadores de la lista

- `<lista>.append(<elemento>)` - Agrega el `<elemento>` al final de `<lista>`
- `<lista>.insert(<índice>, <elemento>)` - Inserta el elemento en `<índice>` y desplaza a la derecha los demás elementos para hacerle espacio (reindexación)
- `<lista>.remove(<elemento>)` - Busca, quita y devuelve al primero `<elemento>` y desplaza a la izquierda los demás elementos (reindexación).
- `<lista>.pop()` - Quita y devuelve al último elemento de `<lista>`
- `<lista>.pop(<índice>)` - Quita y devuelve al elemento de `<índice>` (reindexación)
- `<lista>.clear()` - Quita todos los elementos de `<lista>`

Nota: Al eliminar un `<elemento>` o `<índice>` si no está lista o el índice no es correcto en el rango de la lista se provocará un error.

[ ]:
```
A = []

A, type(A)
```

[ ]:
```
([], list)
```

[ ]:
```
A = [1, 2, 3]

A, type(A)
```

[ ]:
```
([1, 2, 3], list)
```

[ ]:
```
Frutas = ["Manzana", "Pera", "Kiwi", "Mango", "Piña"]

Frutas
```

[ ]:
```
['Manzana', 'Pera', 'Kiwi', 'Mango', 'Piña']
```

1.4.1 Operadores secuenciales

[ ]:
```
edades = [38, 45, 11, 34, 89, 34, 31, 23, 25, 27, 17, 28]
```

[ ]:
```
len(edades)
```

[ ]:
```
12
```

[ ]:
```
min(edades)
```

[ ]: 11

[ ]:
```
max(edades)
```

[ ]: 89

[ ]:
```
sum(edades)
```

[ ]: 402

[ ]:
```
sum(edades) / len(edades)
```

[ ]: 33.5

1.4.2 Operadores de lista

[ ]:
```
calificaciones = []
calificaciones
```

[ ]: []

[ ]:
```
calificaciones.append(8.9)
calificaciones
```

[ ]: [8.9]

[ ]:
```
calificaciones.append(9.1)
calificaciones
```

[ ]: [8.9, 9.1]

[ ]:
```
while len(calificaciones) < 10:
  x = float(input("Dame la siguiente calificación: "))
  calificaciones.append(x)

calificaciones
```

```
Dame la siguiente calificación: 5
Dame la siguiente calificación: 7
Dame la siguiente calificación: 8
Dame la siguiente calificación: 6
Dame la siguiente calificación: 9
Dame la siguiente calificación: 10
Dame la siguiente calificación: 3
Dame la siguiente calificación: 6
Dame la siguiente calificación: 7
Dame la siguiente calificación: 8
```

[ ]:
```
[5.0, 7.0, 8.0, 6.0, 9.0, 10.0, 3.0, 6.0, 7.0, 8.0]
```

[ ]:
```
calificaciones.clear()
calificaciones
```

[ ]:
```
[]
```

1.4.3 Ejemplo: Ordenamiento ingenuo de listas

[ ]:
```
# Entrada
A = [6, 4, 5, 7, 3, 8, 9, 1, 2, 3, 4]

# Inicio
B = []

while len(A) > 0:
  x = min(A)
  A.remove(x)
  B.append(x)

B
```

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

1.5 Índices

Los índices son valores automáticos que usan listas para acceder a los elementos.

Sintaxis para acceder a un índice

```
x = <lista>[<índice>]
```

[ ]:
```
Frutas = ["Manzana", "Pera", "Kiwi", "Piña", "Mango"]

Frutas[0]
```

[ ]:
```
'Manzana'
```

[ ]:
```
Frutas[3]
```

[ ]:
```
'Piña'
```

[ ]:
```
Frutas[-1]
```

[ ]:
```
'Mango'
```

[ ]:
```
Frutas[-4]
```

[ ]:
```
'Pera'
```

1.5.1 Sublistas

Los índices se pueden usar combinados en la estructura `i:j` para extraer todos los elementos desde `i` hasta `j - 1` y devolvemos una sub-lista con esos elementos.

Nota: Podemos ignorar a la `i` o a la `j` y se asumirá el mejor, por ejemplo, si ignoramos a `i` irá desde el principio y si ignoramos a `j` irá hasta el final.

[ ]:
```
Frutas[2:4]
```

[ ]:
```
['Kiwi', 'Piña']
```

[ ]:
```
Frutas[:4]
```

[ ]:
```
['Manzana', 'Pera', 'Kiwi', 'Piña']
```

[ ]:
```
Frutas[2:]
```

[ ]:
```
['Kiwi', 'Piña', 'Mango']
```

Ejecicio: Define una lista con 10 edades y reporta las últimas 3

[ ]:
```
# Solución

edades = [18, 23, 45, 32, 12, 11, 46, 76, 21, 34]

edades[-3:]
```

[ ]:
```
[76, 21, 34]
```

[ ]:
```
edades_ordenadas = sorted(edades)

edades_ordenadas
```

[ ]:
```
[11, 12, 18, 21, 23, 32, 34, 45, 46, 76]
```

[ ]:
```
edades_ordenadas = sorted(edades, reverse=True)

edades_ordenadas
```

[ ]:
```
[76, 46, 45, 34, 32, 23, 21, 18, 12, 11]
```

[ ]:
```
edades_ordenadas[:3]
```

[ ]:
```
[76, 46, 45]
```

[ ]:
```
edades_ordenadas[-3:]
```

[ ]:
```
[18, 12, 11]
```

[ ]:
```
edades_ordenadas[3:-3]
```

[ ]:
```
[34, 32, 23, 21]
```

1.6 Iterador

El iterador es una estructura de control que permite recorrer elementos dentro de una secuencia.

Sintaxis del iterador

```
for <elemento> in <secuencia>:
  <sentencias (elemento)>
```

Repetimos un **bloque de sentencias** por cada *elemento* de la lista o secuencia.

[ ]:
```
Frutas = ["Manzana", "Pera", "Kiwi", "Fresa", "Piña", "Mango"]

for fruta in Frutas:
  print(fruta)
```

```
Manzana
Pera
Kiwi
Fresa
Piña
Mango
```

1.6.1 Ejemplo

Calcular la suma de los números $1 + 2 + 3 + 5 + 8 + 13 + 21$

Esto se puede resolver mediante sumas parciales:

$$S_0 = 0 S_1 = S_0 + 1 S_2 = S_1 + 2 S_3 = S_2 + 3 S_4 = S_3 + 5 S_5 = S_4 + 8 S_6 = S_5 + 13 S_7 = S_6 + 21$$

[ ]:
```
fibos = [1, 2, 3, 5, 8, 13, 21]

s = 0

for x in fibos:
  s = s + x
  print(s)
```

```
1
3
6
11
19
32
53
```

[ ]: 53

[ ]:
```
edades = sorted([18, 23, 17, 19, 21, 34, 85, 14])

s = 0

for edad in edades:
  s = s + edad
  print(s, s / sum(edades))
```

```
14 0.06060606060606061
31 0.1341991341991342
49 0.21212121212121213
68 0.2943722943722944
89 0.3852813852813853
112 0.48484848484848486
146 0.6320346320346321
231 1.0
```