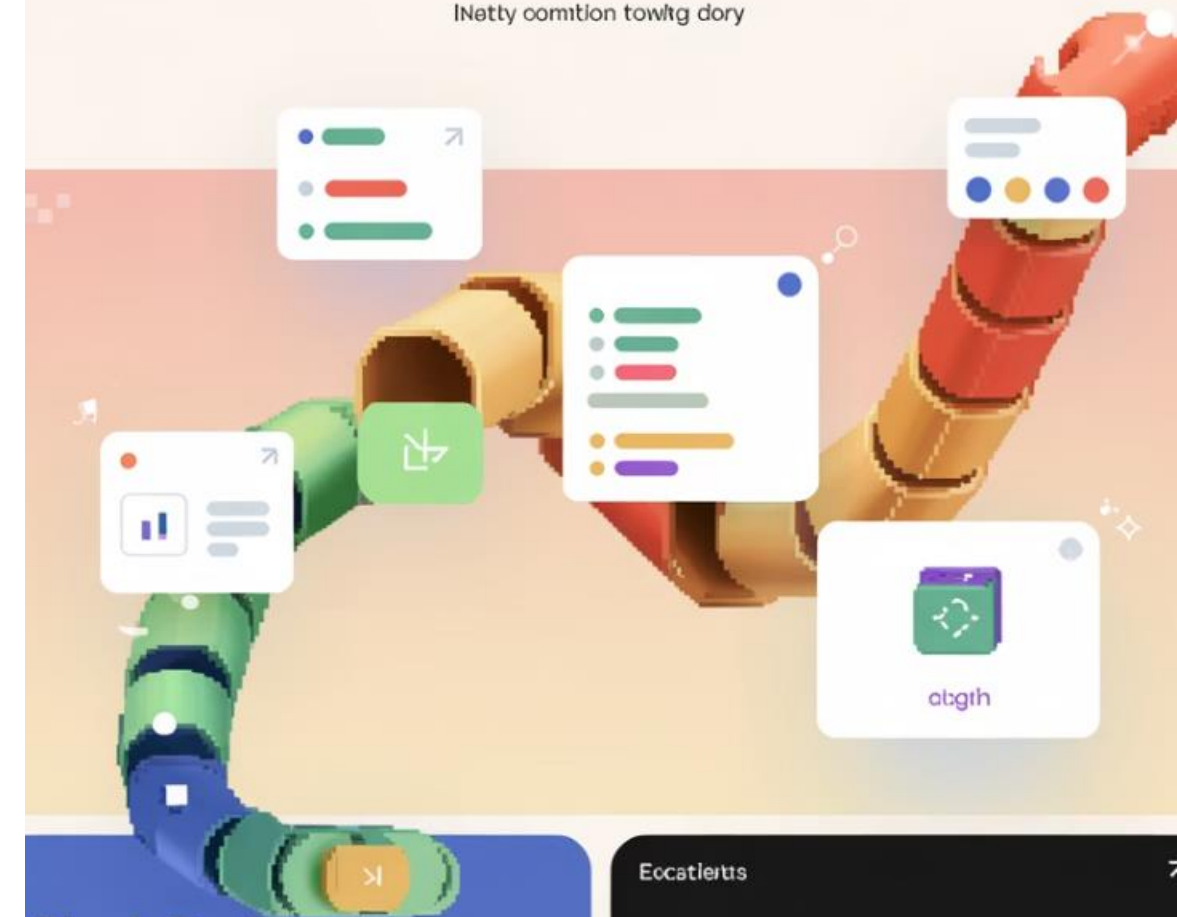


Listas en Python: Estructura de Datos Fundamental

Las listas son una de las estructuras de datos más utilizadas en Python. Permiten almacenar secuencias de elementos de manera ordenada y mutable.

Unlock the power of data visualization with python

INetty comtion towltg dory



Dúsk Dóles with Python

H+D

cretootiele áktuoetínolúíoh úléed póhn,
rnettd táti rífs the pítapclíent,
tínes auslíks tans tíhuésttuólvs
ínrecte xpents cop Tíroel)

Eccatlerits



Creación de Listas

Con corchetes

```
elementos = [1, 2, 3, 4, 5]
```

Con strings

```
nombres = ["Ana", "Luis", "Pedro"]
```

Mixtas

```
mixta = [1, "Hola", 3.14, True]
```

Vacías

```
vacía = []
```

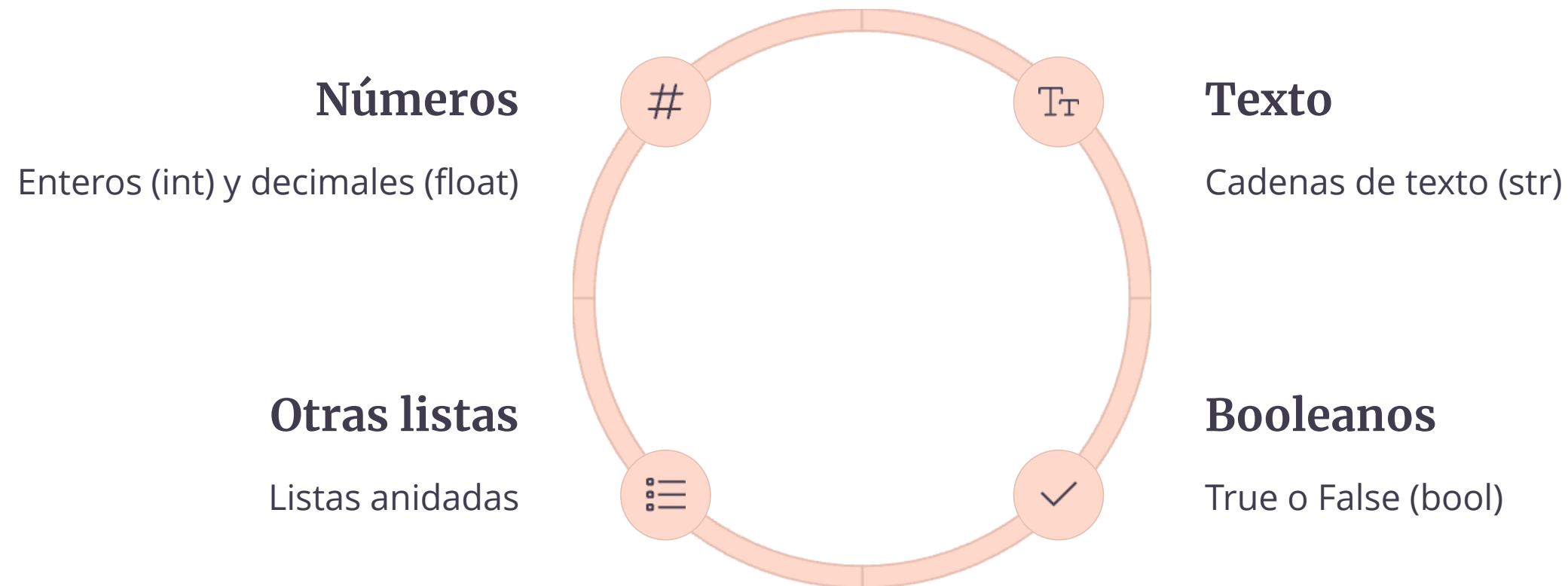


Codespark:
Unleash
your
python
potential

Try it free

Learn more

Tipos de Datos en Listas



Acceso a Elementos

1

Índices positivos

```
nombres = ["Ana", "Luis", "Pedro"]  
print(nombres[0]) # Imprime "Ana"
```

2

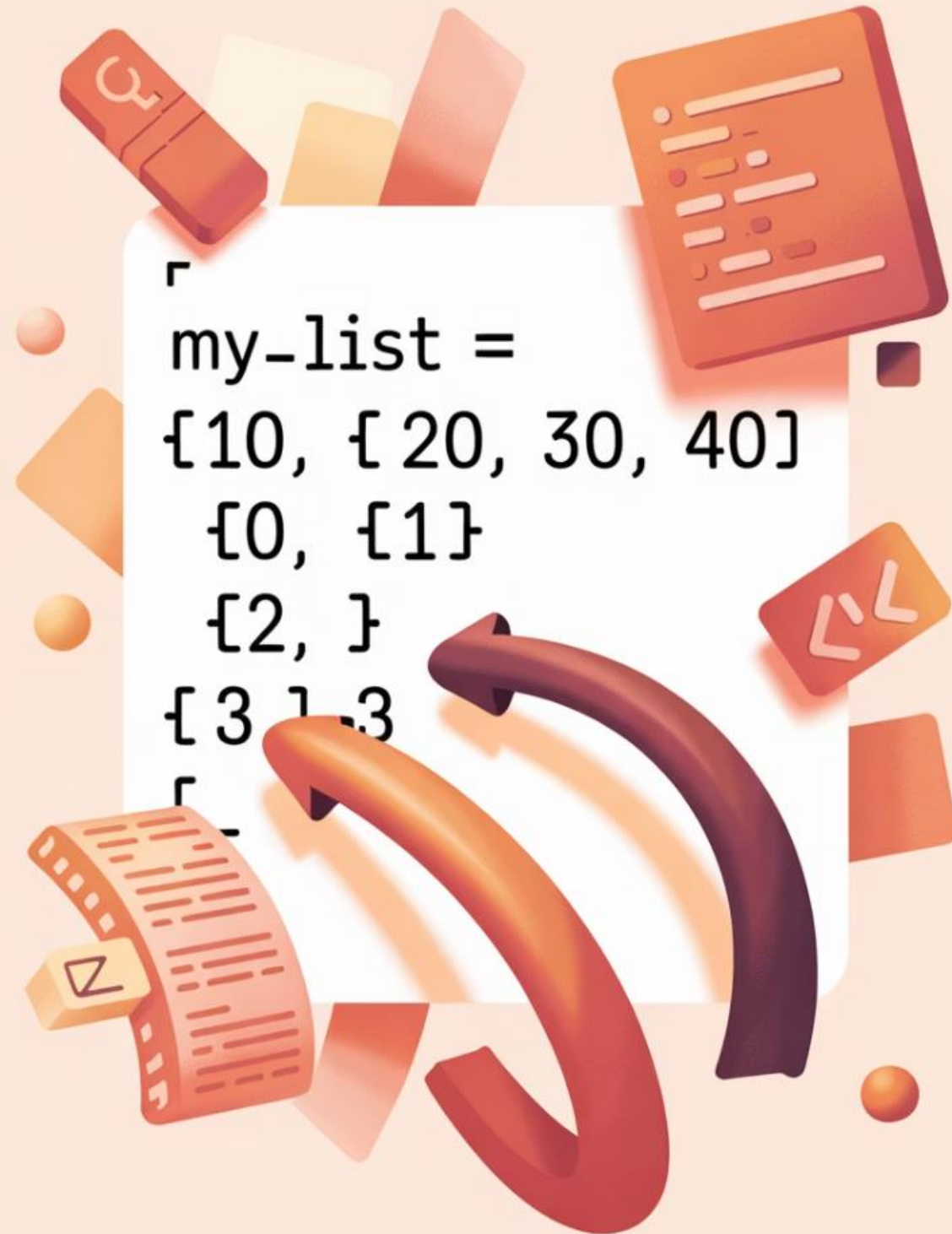
Índices negativos

```
print(nombres[-1]) # Imprime "Pedro"
```

3

Listas anidadas

```
anidada = [[1, 2, 3], [4, 5, 6]]  
print(anidada[0][1]) # Imprime 2
```



Python

Range Examples

```
Fcatyon D-Pyce-i cliag ,costliyg)))
5 ranger-eto--t--sln Fangge)
trotunn,coon 3
a | clcl,ecgnenlien "
l | l toccgne-t ctan,cn,
t | tiogcnent h-h -uncliag)))
e | teague-c-etun,coon,
e | ?
7 | L! t tus ruts"
R | Let(((ecguunc ppatccuclogg
5 | tcclong- ts-counrlieclglug)))
t | clcogrons--? cneencjlug)))
t | trognge--t ttblliccleants,ustliug))-
| tragne--tclttteelelctscLunc,"
L?
b | lcl(3 lounn tcyeliccluts
a | tnung lntun thc cluctyonretiyo))
h, | pactyo -bange--cancangea orscilnr))
? python, turetne, ?
```

Creación con Range()

Secuencia básica

```
numeros = list(range(5))
print(numeros) # [0, 1, 2, 3, 4]
```

Con valor inicial

```
numeros = list(range(2, 10))
print(numeros) # [2, 3, 4, 5, 6, 7, 8, 9]
```

Con paso

```
numeros = list(range(2, 10, 2))
print(numeros) # [2, 4, 6, 8]
```


Python List Slicing

`{start : end:}step`

Slicing de Listas

Rango específico

```
letras = ['a', 'b', 'c', 'd', 'e']  
print(letras[1:4]) # ['b', 'c', 'd']
```

Hasta el final

```
print(letras[2:]) # ['c', 'd', 'e']
```

1

2

3

4

Desde el inicio

```
print(letras[:3]) # ['a', 'b', 'c']
```

Con índices negativos

```
print(letras[-3:-1]) # ['c', 'd']
```

Métodos Split y Modificación

Split: String a Lista

```
texto = "Hola mundo Python"
```

```
palabras = texto.split()
```

```
print(palabras) # ['Hola', 'mundo', 'Python']
```

```
texto2 = "manzana,banana,pera"
```

```
frutas = texto2.split(",")
```

```
print(frutas) # [manzana,banana,pera]
```

Modificación de Listas

```
numeros = [1, 2, 3]
```

```
numeros.append(4) # Añadir al final
```

```
print(numeros) # [1, 2, 3, 4]
```

```
numeros.remove(2) # Eliminar valor
```

```
print(numeros) # [1, 3, 4]
```

Concatenación de Listas en Python

Existen varias maneras de unir o concatenar listas en Python:

Usando el operador +

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
resultado = lista1 + lista2
print(resultado) # [1, 2, 3, 4, 5, 6]
```

Este método crea una nueva lista con los elementos de ambas listas.

Usando el operador *

```
lista = [0] * 4
print(lista) # [0, 0, 0, 0]
```

Este operador permite repetir una lista un número específico de veces.

La elección del método dependerá de si quieres mantener las listas originales o modificarlas.

La palabra clave "in" en Python

En Python, la palabra reservada "in" tiene dos usos principales:

Operador de pertenencia

Permite verificar si un elemento existe dentro de una secuencia (listas, tuplas, strings, etc.).

```
if 'a' in 'Python':  
    print("La letra 'a' está en Python")  
  
numeros = [1, 2, 3, 4, 5]  
if 3 in numeros:  
    print("El número 3 está en la lista")
```

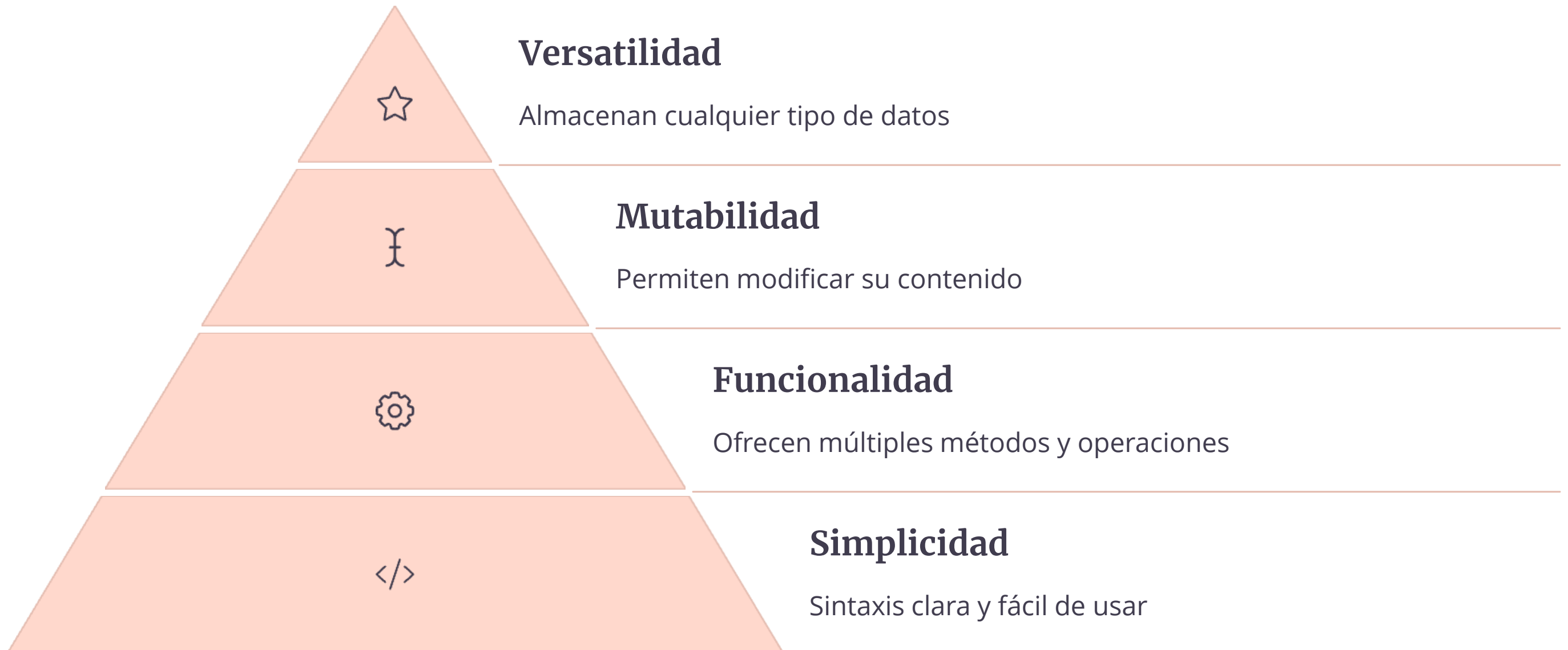
Parte de estructuras de control

Se utiliza en bucles for para iterar sobre elementos de una secuencia.

```
for fruta in ['manzana', 'banana',  
             'naranja']:  
    print(fruta)  
  
for i in range(5):  
    print(i) # Imprime números del 0 al 4
```

El operador "in" devuelve True si el elemento está presente en la secuencia y False en caso contrario, lo que lo hace muy útil para validaciones y filtrado de datos.

Resumen: Poder de las Listas en Python



Ejercicios Prácticos: Listas en Python

1 Creación y Manipulación

Crea una lista de tus cinco frutas favoritas. Luego utiliza `append()` para añadir una sexta fruta y `remove()` para eliminar la segunda fruta de la lista. Imprime la lista resultante.

3 Conversión con `split()`

Convierte la cadena "Python,Java,C++,JavaScript,PHP" en una lista de lenguajes de programación utilizando `split()`. Luego añade "Ruby" a la lista e imprime los lenguajes ordenados alfabéticamente.

2 Slicing Avanzado

Dada la lista `numeros = [10, 20, 30, 40, 50, 60, 70, 80, 90]`, utiliza slicing para obtener: los primeros tres elementos, los últimos dos elementos, y los elementos en posiciones pares.

4 Usando `range()`

Genera una lista con los números impares del 1 al 20 utilizando `range()`. Luego, crea otra lista que contenga sólo los números de la primera lista que son divisibles por 3.

Solución: Creación y Manipulación

El ejercicio pide crear una lista de cinco frutas favoritas, añadir una sexta con `append()` y eliminar la segunda fruta con `remove()`.

```
# Creamos la lista con cinco frutas favoritas
frutas_favoritas = ["manzana", "plátano", "fresa", "mango", "piña"]
print("Lista original:", frutas_favoritas)

# Añadimos una sexta fruta con append()
frutas_favoritas.append("naranja")
print("Después de append():", frutas_favoritas)

# Eliminamos la segunda fruta (plátano) con remove()
frutas_favoritas.remove("plátano")
print("Después de remove():", frutas_favoritas)

# Resultado final
print("Lista resultante:", frutas_favoritas)
# [manzana, fresa, mango, piña, naranja]
```

Observa cómo la lista original se modifica directamente con los métodos `append()` y `remove()`, demostrando la mutabilidad de las listas en Python.

Solución: Slicing Avanzado

El ejercicio pide trabajar con slicing en la lista **numeros = [10, 20, 30, 40, 50, 60, 70, 80, 90]** para obtener diferentes subconjuntos.

```
# Lista original
numeros = [10, 20, 30, 40, 50, 60, 70, 80, 90]
print("Lista original:", numeros)

# Primeros tres elementos: numeros[0:3]
primeros_tres = numeros[:3]
print("Primeros tres elementos:", primeros_tres) # [10, 20, 30]

# Últimos dos elementos: numeros[-2:]
ultimos_dos = numeros[-2:]
print("Últimos dos elementos:", ultimos_dos) # [80, 90]

# Elementos en posiciones pares (índices 0, 2, 4, 6, 8)
posiciones_pares = numeros[::2]
print("Elementos en posiciones pares:", posiciones_pares) # [10, 30, 50, 70, 90]
```

El slicing nos permite extraer fácilmente subsecuencias de una lista usando la sintaxis **lista[inicio:fin:paso]**, donde cualquiera de estos parámetros puede omitirse para usar valores predeterminados.

Solución: Operaciones con Listas

El ejercicio nos pide convertir una cadena de texto en una lista y añadir un elemento.

```
# Cadena original
lenguajes_texto = "Python,Java,C++,JavaScript,PHP"
print("Cadena original:", lenguajes_texto)

# Convertir cadena en lista usando split()
lenguajes = lenguajes_texto.split(",")
print("Lista después de split():", lenguajes)  # ['Python', 'Java', 'C++', 'JavaScript', 'PHP']

# Añadir "Ruby" a la lista
lenguajes.append("Ruby")
print("Lista después de append():", lenguajes)  # ['Python', 'Java', 'C++', 'JavaScript', 'PHP', 'Ruby']
```

Este ejercicio demuestra cómo podemos transformar fácilmente cadenas de texto en listas utilizando el método `split()`, que divide una cadena según el delimitador especificado (en este caso, la coma).

También muestra cómo añadir elementos a una lista existente con el método `append()`, una operación común cuando trabajamos con colecciones de datos dinámicas en Python.

Solución: Operaciones con Listas

El ejercicio nos pide generar una lista de números impares y filtrar los divisibles por 3, utilizando las funcionalidades básicas de Python.

```
# 1. Generar una lista con los números impares del 1 al 20 usando range()
numeros_impares = list(range(1, 21, 2))
print("Números impares del 1 al 20:", numeros_impares)  # [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

# 2. Crear otra lista con los números divisibles por 3 de la primera lista
divisibles_por_3 = []
for numero in numeros_impares:
    if numero % 3 == 0:
        divisibles_por_3.append(numero)
print("Números impares divisibles por 3:", divisibles_por_3)  # [3, 9, 15]
```

Esta solución muestra cómo generar números impares utilizando `range()` con un paso de 2. Luego recorremos esta lista con un bucle `for` y utilizamos una condición simple para identificar y almacenar los números que son divisibles por 3. Este enfoque básico demuestra la claridad y simplicidad que Python ofrece para manipular listas y realizar operaciones de filtrado.