

“Aprendiendo y Mejorando: Soluciones Básicas de Lógica en Python para Entrevistas”

**“Aprendiendo y Mejorando: Soluciones
Básicas de Lógica en Python para
Entrevistas”**

Oscar Farfan



Ordenar lista de números con el algoritmo de burbuja

```
1 import random
2 nums = []
3 for x in range(50):
4     nums.append(random.randint(20,50))
5 print(nums)
6
7 def ordenarAZ(list):
8     cambio = 1
9     while cambio == 1:
10         cambio = 2
11         for i in range(len(list)-1):
12             if list[i] > list[i+1]:
13                 list[i], list[i+1] = list[i+1], list[i]
14                 cambio = 1
15     print(list)
16
17 def ordenarZA(list):
18     cambio = 1
19     while cambio == 1:
20         cambio = 2
21         for i in range(len(list)-1):
22             if list[i] < list[i+1]:
23                 list[i], list[i+1] = list[i+1], list[i]
24                 cambio = 1
25     print(list)
26
27 ordenarAZ(nums)
28 ordenarZA(nums)
```

Este código genera una lista de 50 números aleatorios y los ordena en orden ascendente y descendente utilizando el algoritmo de burbuja. 🤵 Es una manera sencilla de ordenar elementos, pero no es la más eficiente cuando las listas son muy grandes. 🤪

Consejos:

- 💡 **Consejo 1:** Si tienes listas grandes, es mejor usar algoritmos más rápidos, como el **QuickSort** o **MergeSort**. ➡
- ⌚ **Consejo 2:** Puedes experimentar con diferentes tamaños de listas para ver cómo se comporta el algoritmo con más datos. 📈



Imprimir triángulo con caracteres ⚡

```
1 def triangulo(size, char):
2     for s in range(size):
3         print(" "*(size-s-1) + char*(2*s+1))
4 triangulo(5,"*")
5
6 def triangulo2(size, char):
7     row = size-1
8     long = 1
9     for x in range(1,size):
10        print(" "*row + char*long)
11        row -= 1
12        long +=2
13 triangulo2(5,"%")
14
15 def print_triangle(size, character):
16     row = size-1
17     long = 1
18     for x in range(1,size):
19         print(" "*row + character*long)
20         row -= 1
21         long +=2
22 print_triangle(6, "$")
```

Este código imprime triángulos con diferentes caracteres. El tamaño del triángulo se ajusta según el valor de size que se le pase a la función. ⚡ Se utiliza un enfoque simple con bucles for para generar las filas del triángulo. 🎒

Consejos:

- 💡 Consejo 1: Puedes cambiar el carácter con el que se forma el triángulo, por ejemplo, usa #, *, o incluso letras.
- 💡 Consejo 2: Experimenta cambiando el valor de size para ver cómo cambia el triángulo. ¡Hazlo más grande y sorprende a todos! 😊
- 💡 Consejo 3: Prueba con diferentes formas para ver qué tipo de figuras puedes crear, como pirámides o diamantes. 💎

🐱 ¿Quién es el gato más famoso? 🐱

```
1 def find_famous_cat(cats):
2     follows = []
3     names = []
4     famous = []
5     for cat in cats:
6         follows.append(sum(cat["followers"]))
7         names.append(cat["name"])
8
9     maximo = max(follows)
10
11    for i in range(len(follows)):
12        if follows[i] == maximo:
13            famous.append(names[i])
14
15    print(famous)
16
17 find_famous_cat([
18     {
19         "name": "Luna",
20         "followers": [500, 200, 300]
21     },
22     {
23         "name": "Michi",
24         "followers": [100, 300]
25     }
])
```

Este ejercicio te ayudará a encontrar el gato más famoso de una lista de gatos, basándose en el número de seguidores que tienen. 🐾 Compara las cantidades de seguidores de cada uno para determinar quién es el más popular. 🐱

Consejos:

- 💡 Consejo 1: Puedes agregar más gatos con diferentes cantidades de seguidores para hacer el programa más interesante. 🐾
- 💡 Consejo 2: ¿Por qué no hacer que el programa calcule el promedio de seguidores para cada gato? ¡Sería un desafío genial! 🤓

Calcular el promedio de estudiantes

```
1 def get_student_average(students):
2     students_1 = {}
3     list_student = [] # Prmedio individual y nombre
4     class_average = 0 # Promedio total
5     total_alumnos = 0
6
7
8     for alumno in students:
9         alumno_average = (sum(alumno["grades"]) / len(alumno["grades"]))
10        list_student.append({"name":alumno["name"], "average":alumno_average})
11        total_alumnos += 1
12
13        class_average += alumno_average
14
15    class_average = class_average / total_alumnos
16    class_average = round(class_average, 2)
17
18    students_1["class_average"] = class_average
19    students_1["students"] = list_student
20
21    print(students_1)
22
23 get_student_average([
24     {
25         "name": "Pedro",
26         "grades": [90, 87, 88, 90],
27     },
28     {
29         "name": "Jose",
30         "grades": [99, 71, 88, 96],
31     },
32     {
33         "name": "Maria",
34         "grades": [92, 81, 80, 96],
35     },])
36
```

Este ejercicio calcula el promedio de notas de un grupo de estudiantes y también te dice el promedio de la clase. 🎓 Es útil para conocer el rendimiento general de un grupo. 🏫

Consejos:

- 💡 Consejo 1: Si quieres hacer más complejo el ejercicio, agrega una condición para aprobar o reprobar dependiendo del promedio. 📊
- 💡 Consejo 2: ¡Anímate a hacer gráficos con los promedios para visualizar mejor los resultados! 📈



Recopilar información sobre paquetes



```
1 def get_packages_info(packages):
2
3     my_dict = {}
4     paises = {}
5     peso_total = 0
6
7     total_packages = len(packages)
8
9     for package in packages:
10         peso = package[1]
11         peso_total += peso
12
13     for package in packages:
14         pais = package[2]
15         paises[pais] = paises.get(pais,0)+1
16
17
18     peso_total = round(peso_total,2)
19     my_dict["total_weight"] = peso_total
20     my_dict["destinations"] = paises
21
22     return print(my_dict)
23
24
25     get_packages_info([
26         (1, 20, "Mexico"),
27         (2, 15.5, "Colombia"),
28         (3, 30, "Mexico"),
29         (4, 12, "Argentina"),
30         (5, 8.2, "Colombia"),
31         (6, 25, "Mexico"),
32         (7, 18.7, "Argentina"),
33         (8, 5, "Colombia"),
34         (9, 22.3, "Argentina"),
35         (10, 14.8, "Colombia")
36     ])
```

Este código recopila información sobre los paquetes, como el peso total y la cantidad de paquetes enviados a diferentes destinos. 🌎 Ideal para sistemas de envíos o logística. 🚚

Consejos:

- 💡 Consejo 1: Prueba agregar más destinos y ver cómo cambia la salida. ¡Haz que los países aumenten! 🌎
- 📊 Consejo 2: Podrías crear un gráfico para mostrar los destinos más populares con un buen análisis de los paquetes. 📈



Contar cuántas veces aparece cada letra



6.letrasOracionDiccionario.py > ...

```
1 def count_letters(phrase):  
2     return print({char: phrase.count(char) for char in phrase})  
3  
4 count_letters("Oscar Esta AQUI hoy Martes")  
5  
6 def count_letters(phrase):  
7     my_dict = {}  
8  
9     for p in phrase:  
10        my_dict[p] = my_dict.get(p,0)+1  
11  
12    return print(my_dict)  
13  
14 count_letters("Hola Mundo")  
15
```

Este código cuenta cuántas veces aparece cada letra en una frase. 🧐 Es útil para análisis de texto y estadísticas sobre letras en cadenas de caracteres. AB
CD

Consejos:

- 💡 Consejo 1: Si quieres complicarlo un poco, agrega un filtro para que ignore los espacios o caracteres especiales. ✖️
- 💡 Consejo 2: Prueba con frases más largas o palabras interesantes como "supercalifragilisticexpialidocious" 😊

💡 Comprobar si una palabra es un palíndromo 🤖

```
1 def palindromo(palabra):
2     palindromo = False
3
4     if palabra == palabra[::-1]:
5         palindromo = True
6
7     return print(f"La palabra {palabra} tiene el estado de {palindromo} en la categoria de palindromo")
8
9
10 palabra = input("ingresa una palabra: ")
11 palindromo(palabra)
12
13
14
```

Este código verifica si una palabra es un palíndromo, es decir, si se lee igual de adelante hacia atrás. 🕵️ Las palabras como "radar" o "madam" son ejemplos de palíndromos. 🎖️

Consejos:

- 🕵️ **Consejo 1:** Experimenta con frases y palabras más largas, ¿será que todo el mundo sabe lo que es un palíndromo? 🤔
- 🤔 **Consejo 2:** ¿Por qué no crear una función que ignore los espacios o los signos de puntuación para hacerla más flexible? 😊

La lógica detrás de los datos

Este proyecto fue más que resolver ejercicios de lógica. Fue un recordatorio de que **la programación no solo trata de escribir código**, sino de **entender el pensamiento detrás de cada instrucción**. 

La **lógica de programación** es el lenguaje que nos permite traducir problemas complejos en soluciones claras y eficientes. Y cuando la combinamos con herramientas de análisis de datos, logramos algo poderoso: transformar números en conocimiento. 

En un mundo impulsado por los datos, **saber cómo pensar es tan importante como saber qué herramienta usar**.

¡Porque el verdadero poder está en conectar la lógica con la interpretación! 

Oscar Farfan | Analista de Datos