Aqui está uma **lista de 30 exercícios avançados** em Python, especialmente projetados para consolidar e desafiar os conceitos de:

- **Strings** (métodos avançados, fatiamento inteligente, `len()`, `in`, `find()`, `count()`, `replace()`, `split()`, `join()`, etc.)
- **Condicionais complexos** (`if/elif/else` com múltiplos operadores lógicos `and/or/not`, expressões compostas)
- **Laços `while`** (contadores, validação robusta de entrada, loops infinitos com break/continue, controle fino de fluxo)

Os exercícios são **progressivamente mais difíceis** e combinam os três tópicos na maioria dos casos (o que torna cada um mais "avançado" e próximo de problemas reais).
Ideal para alunos que já dominam o Nível 1 básico e querem subir de nível.

Exercícios 1–10: Strings + Condicionais Avançados
1. Receba uma frase e determine se ela é um **pangrama** (contém todas as letras do alfabeto português, ignorando acentos, maiúsculas e espaços). Use `in` e condicionais.

2. Receba uma palavra e verifique se ela é um **anagrama** de "python" (use apenas métodos de string e condicionais, sem `sorted`).

3. Crie um validador de **senha forte** que só aceite a senha se tiver:
   - mínimo 8 caracteres
   - pelo menos 1 letra maiúscula, 1 minúscula, 1 número e 1 símbolo
   Use `while` + condicionais compostas.

4. Receba um texto longo e substitua **todas as palavras com 4 ou mais letras** por asteriscos do mesmo tamanho (ex: "segredo" → "*******").

5. Verifique se uma string é um **número de telefone brasileiro válido** (formato (XX) XXXXX-XXXX ou (XX) XXXX-XXXX). Use fatiamento e múltiplas condições.

6. Receba uma frase e retorne a versão **alternada maiúscula/minúscula** (ex: "Python é incrível" → "PyThOn É iNcRíVeL").

7. Implemente um **detector de palíndromo ignorando acentos, espaços e pontuação**. Ex: "A sacada da casa" deve retornar True.

8. Receba um CPF (apenas números) e valide se é um CPF **matematicamente válido** (usando a regra dos dígitos verificadores). Use fatiamento e condicionais.

9. Crie um programa que conte quantas **substrings palíndromas** de tamanho ≥ 3 existem em uma string (ex: "abaaba" tem várias).

10. Receba um texto e retorne quantas palavras começam e terminam com a **mesma letra** (maiúscula ou minúscula).

Exercícios 11–20: While + Validação + Strings
11. Peça ao usuário que digite um e-mail até que ele seja válido (deve conter "@" e "." depois do "@"). Use `while` + validação com `in` e `find()`.

12. Simule um **caixa eletrônico**: peça o valor do saque (múltiplos de 10) e só aceite quando for válido. Mostre quantas tentativas foram necessárias.

13. Peça uma senha e só permita prosseguir quando ela for diferente da senha anterior (guarde a última senha em uma variável). Máximo 5 tentativas.

14. Receba números como string até que o usuário digite “fim”. Valide que cada número tem exatamente 4 dígitos e é par. Conte quantos foram válidos.

15. Crie um **validador de data** (formato DD/MM/AAAA). Use `while` até o usuário digitar uma data válida (verifique dias por mês, ano bissexto com condicionais).

16. Peça uma frase e, enquanto o usuário não digitar uma frase com **pelo menos 3 palavras diferentes**, continue pedindo.

17. Implemente um contador de **tentativas de login**: o usuário tem 3 chances de acertar “admin123”. Após 3 erros, bloqueie por 5 segundos simulados (`import time` se quiser, mas não obrigatório).

18. Receba strings até o usuário digitar “SAIR”. Para cada string, verifique se contém a palavra “python” (case insensitive) e conte quantas continham.

19. Crie um loop que peça um código de produto (formato XXX-999) até que seja válido. Valide com fatiamento e condicionais compostas.

20. Peça um texto e, enquanto o texto tiver **menos de 50 caracteres ou mais de 200**, continue pedindo. Depois mostre quantas vogais (com acento) ele tem.

Exercícios 21–30: Integração Avançada (Strings + Condicionais + While)
21. Simule um **jogo de adivinhação de palavra**: o programa escolhe uma palavra secreta (“python”). O usuário tem 8 tentativas de chutar letras. Mostre a palavra com _ e letras acertadas.

22. Crie um **analisador de força de senha** em loop: o usuário digita senhas até conseguir uma nota 100/100 (pontue com condicionais complexas: comprimento, variedade de caracteres, etc.).

23. Receba múltiplas frases até o usuário digitar “fim”. Para cada frase, diga se ela é **pergunta** (termina com “?”), **exclamação** ou **afirmação**. Conte o total de cada tipo.

24. Implemente um **mini corretor ortográfico** simples: peça palavras e, se a palavra estiver na lista proibida [“nao”, “vc”, “pq”], sugira a forma correta. Use `while` até o usuário digitar “ok”.

25. Valide uma **sequência de DNA** (apenas A, T, C, G). O usuário pode digitar várias sequências até digitar “fim”. Mostre o percentual de GC para cada sequência válida.

26. Crie um programa que leia números como string e some **apenas os dígitos pares** de cada número digitado. Pare quando o usuário digitar um número negativo.

27. **Desafio de criptografia**: implemente uma cifra de César (deslocamento 3) que só criptografa se a frase tiver pelo menos 10 caracteres e conter pelo menos uma letra maiúscula. Use `while` para pedir até atender os critérios.

28. Peça ao usuário que digite uma **expressão matemática simples** (ex: “23+45*2”) e valide se contém apenas números, +, -, *, / e parênteses. Se válida, avalie com `eval()` (ou implemente manualmente com condicionais).

29. Crie um **filtro de palavrões**: o usuário digita frases até digitar “sair”. Para cada frase, substitua 5 palavrões comuns por “***” e mostre a frase censurada. Conte quantas palavras foram censuradas no total.

30. **Projeto integrador**: Simule um **sistema de cadastro de usuários**.
- Peça nome (mínimo 3 letras), e-mail válido, senha forte.
- Use `while` para validar cada campo individualmente.
- Ao final, mostre “Cadastro aprovado” apenas se todos os 3 campos estiverem corretos.