Muito fácil 1

Importei pi da biblioteca math e usei a formula de conversão de radianos para graus , e usei a função round para delimitar quantas casas após a virgula queria

Muito fácil 2

Fiz as relações matemáticas entre as hora e as outras medidas de tempo e converti os valores quebrados de mês e semana usando {:.0f}

Fácil 1

Usei um for de y a range n para colocar a palavra na lista criada na função

Fácil 2

Usei a função do python bin que transforma um número inteiro uma str com seu número binário , depois foi só fazer um for com range do tamanho da str e colocar um contador para todas as vezes que o valor 1 aparecesse

Médio 1

Importei da biblioteca cmath a função sqrt e fiz a raiz do quadrado da diferença dos valores em x mais o quadrado da diferença dos valores em y .

Médio 2

Criei uma sequência de ifs para primeiramente analisar se o número era divisível por 3 e por 5, em seguida se o número e divisível por 5 e depois por 3, retornando seus respectivos valores.

Médio 3

Nunca tive contato com programação com sensores em tempo real, então não consegui fazer um método eficiente para analisar as distancias laterais do robô, simplesmente segui os tutoriais do site e fiz ele andar aleatoriamente na pista sem cair, aprendi um pouco sobre a construção do robô, mas tudo que eu mexia o robô saia voando, empinava, pesava 10 kg ou ficava rodando sem parar. A única coisa que fiz diferente dos tutorias do site que não deu tão ruim foi colocar o robô pra virar se a alguma distancia lateral fosse 0, mas não tenho ideia de como identificar e atacar o inimigo, ainda.

Muito Difícil 1

Primeiramente criei a função de criar uma matriz normal e condicionei ela a ser quadrada e ter de 4 a 7 linhasxcolunas , usei a lógica que o caminho mais perto para chegar na posição desejada era condicionar os movimentos priorizando descer e ir para a direita e só depois subir ou ir para esquerda , então criei a sequência de ifs nessa sequência junto a condicional que a última ação nunca poderia ser o contrário da atual , por exemplo se a última ação foi ir para a cima a ação atual não poderia ser ir para baixo , me deparei com o problema do limite da lista e seus índices , portando adicionei ao final de todas as linha o valor ‘x’ e a primeira e última linha da matriz com valores ‘x’ para que a

referência nunca chegue na extremidade da matriz , chegando assim em resultados corretos , porem pode haver situações de matrizes que meu código da erro de índice , não tenho a certeza se cobri todas as possibilidades , pois pensei em situações que o índice trava ao tentar ir para a esquerda na primeira colunas , mas a condicional pra esquerda é a ultima então não sei ao certo . E se o loop chegar a 1000 tentativas ele é quebrado e a resposta para o sistema é SI.

Muito Difícil 2

Como eu não tinha a menor ideia matemática ou esquemática de como resolver um sudoko sistematicamente , separei todas as linhas , colunas e quadrantes em listas separadas , minha primeira ideia foi seguir as linhas em um for e avaliar , se o número fosse 0 eu pegaria os números sequenciais de 1 a 9 e verificaria se eles não estavam na linha no quadrante ou na respectiva coluna , essa primeira ideia se tornou a versão 1 do programa , onde era impossível chegar a um resultado pois a linha estava condicionada a uma sequência numérica que fazia a ordem estar errada e vários espaços continuarem com o valor 0 , também na primeira versão esqueci de atualizar as colunas e quadrantes , mas visto o problema pensei em uma maneiras de completar totalmente uma linha sem os valores 0 , então criei a função solve que resolvia a linha com valores aleatórios e se no final algum valor ainda fosse 0 ela tentaria de novo até chegar no resultado de uma linha com 9 números distintos ,diferentes de 0 e condicionalmente relacionados ao sudoko, essa ideia virou a versão 2 do código , que tinha o problema de apenas entregar uma possível versão para apenas uma linha , gerando loops infinitos quando a sequência de linhas impossibilitava a continuidade do código , portanto usei uma função para reiniciar o código sempre

que o loop infinito ocorresse e coloquei dentro da função solve , fui alterando os valores das colunas e quadrantes e condicionando sempre com a próxima linha , e se entrasse em loop infinito o programa era reiniciado , e em meio a aleatoriedade consegui chegar ao resultado , acontecendo de primeira até ao erro OSError: [Errno 12] Not enough space , que não sei exatamente o que significa, mas se o código demorar muito para achar a solução alguma memoria em algum lugar fica cheia , sendo que em 30 testes consecutivos 9 apresentaram esse erro e os outros 21 em uma solução possível para o problema. Talvez eu tenha resolvido o problema, tirei todos os prints que eu deixei marcado com # , que mostravam os passos que o código estava e uma mensagem de jaja da boa, e em 30 testes consecutivos o erro não voltou, mas como não sei o motivo dele estar lá, não tenho ideia do motivo de ter sumido, pois como se trata de experimentos aleatórios posso ter dado sorte nos 30, mas gostei da porcentagem. Descobri que só dei sorte mesmo porem, resolvi esse problema, hoje fui tentar fazer uma interface gráfica por qtdesing para o sudoko porem como estava usando uma função baseada em comando os e sys para reiniciar o programa toda vez que tentava solucionar o sudoku, a janela era reiniciada, então tive a ideia de fazer a função restart_program simplesmente chamar a função sudoko novamente , ideia que funcionou mas me trouxe ao problema do programa no terminal não ser finalizado após o print do sudoku resolvido , porem o erro OS não apareceu mais , então desisti da interface e de retornar um valor , coloquei um exit() no final do código e em 150 testes nenhum problema surgiu , tornando a limitação do código ser que não há um return e sempre o código será finalizado após o resultado ser encontrado e printado no terminal, impossibilitando o uso de dados obtidos após o uso da função.

Dificil 3

Achei na internet a simulação de monte carlos e fiz uma versão simplificada dela sem usar plotagem, a ideia se baseia que se você pega pontos com o limite ao infinito em um círculo divido pela quantidade de pontos no quadrado vezes 4 é igual a pi, então eu peguei n números aleatórios de 0 a 1 e vi se a distância do valor até o centro (0.5,0.5) era menor que 0.5 que é o raio do círculo hipotético, e coloquei um contador nesse for, depois fiz a conta de 4*contador/n.