DOCUMENTAÇÃO

De: Felipe Sancho

Aluno n.: ist1109611

Âmbito: Hackerschool

Assunto: Documentação do projeto de Python

Data: 4 de janeiro de 2024

Conteúdo

1 Overview

Este projeto consiste num horário, no qual se podem adicionar e remover tarefas, de modo a que estas fiquem ordenadas pelo tempo de início de cada uma. O horário é persistente entre execuções do programa.

Para executar este programa é apenas necessária uma instalação de Python. Para o desenvolvimento foi utilizada a versão 3.12.1.

O projeto foi organizado com respeito ao paradima de programação por camadas, de modo a garantir a independência de dados, sendo que cada módulo definido e utilizado no programa corresponde a uma camada.

2 Descrição das camadas

Nesta secção serão descritos os métodos de cada camada necessária para a execução do programa, e também como foi definido cada método. É importante destacar que devido á utilização do paradigma de programação por camadas estas definições não são as únicas possíveis. Desde que as camadas tenham os mesmos métodos e estes tenham os mesmos ”inputs”e ”outputs”as suas definições podem ser feitas de muitas maneiras diferentes. As definições utilizadas também não são particularmente eficientes e podem com certeza ser melhoradas.

2.1 CAP

A CAP ou ”Cadeia de Acontecimentos Pendentes”é a base do programa. Esta camada é a agenda que mantém as tarefas em ordem cronológica.

2.1.1 Métodos implementados

new() Cria uma nova cadeia

add(cap, evt) adiciona o evento *evt* à cadeia *cap* mantendo a ordem cronológica

remove(cap, Kind, Time) remove da cadeia *cap* o evento do tipo *Kind* que está a ocorrer no dado tempo *Time*. Se mais de um evento do mesmo tipo estiverem a ocorrer ao mesmo tempo são os dois removidos da cadeia.

2.1.2 Implementação

A CAP está implementada como uma lista de eventos ordenada por ordem cronológica.

new Simplesmente retorna uma lista vazia

add Faz uma pesquisa binária na cadeia k*cap* pelo tempo de inicio de cada evento e adiciona o evento *[*evt] quando encontrar a posição onde deve estar utilizando o método insert no índice da lista que resultou do processo de pesquisa binária.

remove Utiliza o combinador filter para manter todos os elementos que não são do tipo *Kind* ou que o tempo *Time* dado não está no intervalo definido pelo tempo de início de tempo de fim do evento $[events.Stime(evt), events.Ftime(evt)[$. Ou seja, remove apenas os eventos do tipo *Kind* e que o tempo *Time* está incluído no intervalo anterior.

2.2 events

Os eventos ou "tarefas"são os acontecimentos que são adicionados ao gerenciador de tarefas. Cada evento tem 4 atributos que são, Tipo(Kind), Descrição(Desc), Tempo de início(Stime) e Tempo de fim(Ftime). Ftime tem que ser obrigatóriamente maior que Stime.

2.2.1 Métodos implementados

new((kind, desc, Stime, Ftime) cria um novo evento com os atributos dados

kind (evt) Retorna o tipo do evento *evt* dado

desc (evt) Retorna a descrição do evento *evt* dado

Stime (evt) Retorna o tempo de início do evento *evt* dado

Ftime (evt) Retorna o tempo de fim do evento *evt* dado

2.2.2 Implementação

Os eventos estão de forma simples como uma tupla de 4 elementos ordenados, do tipo **(kind, desc, Stime, Ftime)**

new retorna uma tupla (kind, desc, Stime, Ftime)

kind retorna o primeiro elemento da tupla *evt*

desc retorna o segundo elemento da tupla *evt*

Stime retorna o terceiro elemento da tupla *evt*

Ftime retorna o quarto elemento da tupla *evt*

3 Programa

O programa principal é executado a partir do ficheiro *main.py*.

3.1 Setup e persistência

O primeiro passo do programa é importar os módulos necessários e criar um novo horário.

A persistência é atingida através do módulo *json*. O próximo passo do programa é verificar o tamanho do ficheiro "persistance.json". Se o tamanho deste for maior do que 2 - o que implica que existe um horário guardado - este é carregado como o horário atual.

Para manter a independência das camadas recorre-se a um processo pouco eficiente para transformar o horário guardado num horário novo. Isto é necessário pois na definição da camada *events* utilizada, as tarefas são definidos como tuplas, que quando são guardadas no ficheiro .json são transformadas em listas. Este processo garante que esta definição se mantém quando é carregado um horário guardado.

Caso contrário, o horário continua vazio.

```
import CAP,events,json,os
#criar/pegar o horário

H = CAP.new()
if os.path.getsize("persistance.json") > 2:
    with open("persistance.json","r") as f:
        p = json.load(f)

        while CAP.size(p) != 0:
            evt = CAP.next(p)
            H = CAP.add(H, events.new(events.kind(evt),events.desc(evt),events.Stime(evt),events.Ftime(evt)))
            p = CAP.remove_first(p)
```

3.2 pretty_print

A função pretty_print é definida neste ponto. A utilidade desta função é imprimir o horário de uma forma mais legível, isto é conseguido através ciclo while e uma função auxiliar, imprimindo cada tarefa do horário numa linha do terminal, pela ordem (Tipo, Descrição, Tempo de início, Tempo de fim).

```
#Função utilizada ao longo do programa
def pretty_print(H):
    aux = H[:]
    print("Horário Atual:")
    while CAP.size(aux) != 0: #Imprime o horário em um formato mais legível
        print(CAP.next(aux))
        aux = CAP.remove_first(aux)
```

3.3 Main loop

O loop principal no programa é um while True. O programa tem uma interface simples no terminal feita através de comandos *input* para determinar qual a escolha do utilizador, e comandos *if* para executar certas partes do programa, dependendo da escolha. Todas as opções do programa ocorrem a partir de comandos *if*. Se o número colocado pelo utilizador não for nenhum dos números definidos para escolhas, o programa simplesmente pede para introduzir um novo número.

3.3.1 Adicionar uma nova tarefa

Neste caso, o programa pede o tipo, a descrição, o tempo de início e o tempo de fim da tarefa a ser adicionada. Para impedir o utilizador de introduzir uma tarefa que tenha um tempo de início após o seu tempo de fim, foi implementado um pequeno ciclo while True, que caso isso aconteça, pede para o utilizador introduzir os parâmetros novamente.

Então, através do método **add** da camada **CAP** adiciona-la ao horário atual, e por fim imprime o horário com a função pretty_print.

```
if escolha == "1":
    kind = input("tipo do evento: ")
    desc = input("desc do evento: ")
    while True:
        Stime = float(input("tempo de início do evento: "))
        Ftime  = float(input("tempo de fim do evento: "))
        if Stime > Ftime:
            print("O tempo de fim da tarefa deve ser após o tempo de início")
        else:
            break
    H = CAP.add(H, events.new(kind, desc, Stime, Ftime))#adiciona o evento ao horário
    pretty_print(H)
```

3.3.2 Remover uma tarefa

Neste caso, o programa pede o tipo da tarefa a ser removida e o tempo em que está a decorrer. Este tempo pode ser um *floating point number* entre o tempo de início e o tempo de fim da tarefa. Depois, o programa imprime o horário com a função pretty_print.

```
elif escolha == "2":
    kind = input("tipo do evento: ")
    Time = float(input("tempo em que o evento esta a decorrer: "))
    H = CAP.remove(H, kind, Time)
    pretty_print(H)
```

3.3.3 Verificar o horário atual

Esta opção é a mais simples, e apenas executa a função pretty_print no horário.

```
elif escolha == "3":
    pretty_print(H)
```

3.3.4 Limpar completamente o horário, e apagar o horário persistente

Neste caso, o programa define o horário atual como um horário vazio, e guarda-o no ficheiro ”persistance.json”de modo sobrepor o horário atualmente guardado. Fica então um horário vazio guardado.

```
elif escolha == "4":#Limpar o Horário
    H = CAP.new()
    with open("persistance.json","w") as f:
        json.dump(H,f,indent=2)
    print("Horário apagado :(")
```

3.3.5 Sair

Neste caso, o programa pergunta ao usuário se quer guardar o horário atual. O método *lower()* é utilizado para verificar se a escolha do utilizador é ”S”ou ”s”. Caso seja uma destas duas opções o programa guarda o horário atual no ficheiro ”persistance.json”para que este possa ser carregado da próxima vez que o programa é executado.

Então ocorre um *break* para parar o loop principal, encerrando assim o programa.

```
elif escolha == "5":
    print("Deseja guardar o horário atual?")
    guardar = input("S/N: ")
    if guardar.lower() == "s":
        with open("persistance.json","w") as f:#Guarda o horário no ficheiro Json
            json.dump(H,f,indent=2)
    break
```

4 Conclusão

Isto conclui a documentação do programa desenvolvido para o projeto de python. :)