Especificação do Projeto de Programação 1 (IF968)

Fernando Castor
Centro de Informática
Universidade Federal de Pernambuco

1 de junho de 2017

1 Introdução

O objetivo deste trabalho é praticar a escrita de funções e programas em Python, em particular, programas envolvendo strings, vetores, listas, tuplas, dicionários e arquivos. Além disso, é a primeira oportunidade que os alunos têm, no contexto do curso de Sistemas de Informação, de desenvolver um sistema não-trivial, ainda que simples.

Você deve gastar pelo menos uma hora lendo este documento para se certificar de que entendeu completamente o que é pedido, de forma a não perder tempo com idas e vindas ao documento e ter certeza de que de fato está entregando o que está especificado. Leia atentamente o documento destacando partes importantes, anotando as dúvidas para saná-las o quanto antes.

Ao longo do documento, as tarefas que você **tem que** realizar são indicadas da seguinte maneira:

Tarefa 0: *ler o documento com bastante cuidado.*

2 Objetivo

Neste projeto, você deverá desenvolver uma agenda de compromissos para ser usada na linha de comando. A principal inspiração para o projeto é o formato padronizado para criar arquivos `todo.txt` disponível em `http://todotxt.com`, combinada com o cliente para linha de comando todo.txt CLI, que gerencia listas de tarefas no formato mencionado.

A agenda fica guardada em um arquivo texto chamado `todo.txt`. Esse arquivo guarda uma lista de afazeres com algumas informações a mais, definidas de maneira padronizada. Seu objetivo neste projeto é construir um programa que manipula esse arquivo para gerenciar seus compromissos de maneira ágil. Milhões de pessoas guardam seus compromissos em arquivos texto como esse e vários milhares especificamente o fazem como será descrito na próxima seção (o professor da disciplina, inclusive).

Os arquivos dos quais você vai precisar para desenvolver o projeto estão disponíveis na página da disciplina. Você precisará de apenas dois:

1. `todo.txt`, um exemplo ultra-simplificado de agenda de compromissos.
2. `agenda.py`, um esqueleto de implementação para o projeto.

Tarefa 1: *Obtenha os arquivos todo.txt e agenda.py a partir do endereço*
`https://sites.google.com/a/cin.ufpe.br/if968si/projeto`

3 O Formato do `todo.txt`

As atividades registradas no arquivo `todo.txt` devem estar organizadas usando um formato padronizado. Esta seção define esse formato que deverá ser usado por seu programa. O trecho abaixo apresenta um exemplo de possível conteúdo para um arquivo `todo.txt` contendo uma lista de afazeres:

```
Comer menos açúcar
23052017 1030 Reunião com Huguinho, Luizinho e Zezinho. @Skype +Pesquisa
(B) Terminar a especificação do projeto de IF968. +IF968
(B) Publicar a especificação do projeto de IF968 terminada. +IF968
(A) Corrigir provas de IF968 +IF968
23052017 (A) Ler artigo IST. +Serviço
2330 Dormir cedo @Casa
```

Cada linha do arquivo acima apresenta uma atividade:

- Cada atividade tem uma **descrição**. Essa descrição pode ser a única informação disponível sobre a atividade. É o caso da primeira atividade, `‘‘Comer menos açúcar’’`.
- A segunda está organizada de forma diferente. Antes da descrição, aparecem uma **data** (`23052017` – 23 de maio de 2017) e uma **hora** (`1030` – 10h30m). Uma atividade que tem essas duas informações é uma que tem data e hora para ser realizada. Tanto hora quanto data são opcionais. A última atividade da lista tem hora mas não data enquanto a penúltima tem data mas não tem hora. A primeira não tem nenhum dos dois.
- No final da segunda atividade aparecem dois pedaços adicionais de informação. O primeiro deles é o **contexto** da atividade (`’@Skype’`, indicado por começar com `’@’`). O contexto indica onde a atividade será realizada. Nesse exemplo, `‘‘@Skype’’` indica que a atividade será conduzida por vídeo conferência, usando o programa Skype. A última atividade do exemplo tem como contexto `‘‘@Casa’’`.
- A última informação que aparece na segunda atividade é o **projeto** ao qual está relacionada, (`+Pesquisa`, indicado por começar com `’+’`). O projeto é uma forma de agrupar atividades relacionadas. No exemplo, há três projetos relacionados à disciplina de Programação 1, cujo código é IF968. Todas essas atividades são marcadas como pertencendo ao projeto `‘‘+IF968’’`.
- Por fim, a terceira, quarta e quinta atividades começam com uma **prioridade**, uma letra de `A` (mais importante) a `Z` (menos importante) que aparece entre parênteses e indica a importância da atividade.

De forma geral, uma atividade tem sempre o seguinte formato:

```
DDMMAAAA HHMM (PRI) DESC @CTX +PROJ
```

onde `DDMMAAAA` é uma data que consiste obrigatoriamente em dia (dois dígitos), mês (dois digitos) e ano (quatro dígitos), `HHMM` é uma hora que consiste obrigatoriamente em hora (dois dígitos) e minutos (dois dígitos), `(PRI)` é uma prioridade que consiste em exatamente uma letra entre `A` e `Z` aparecendo entre parênteses, `DESC` é a descrição de uma atividade, um string com um ou mais caracteres diferentes de espaço em branco, `@CTX` é o contexto de uma atividade, uma cadeia de 2 ou mais caracteres cujo primeiro caractere é `’@’`, e `+PROJ` é o projeto do qual aquela atividade faz parte, uma cadeia de 2 ou mais caracteres cujo primeiro caractere é `’+’`. Como mencionado antes, **todos os elementos de uma atividade exceto pela descrição são opcionais.**.

Tarefa 2: *Crie manualmente um arquivo* `todo.txt` *e inclua nele diversos compromissos como os apresentados nesta seção. Invente novos compromissos, porém, para ter um arquivo que lhe ajude a testar todas as funcionalidades do programa que você vai construir. Seu arquivo também deve incluir algumas atividades fora do formato especificado, por exemplo, com uma data com menos que 8 dígitos, para verificar como seu programa se comporta. O número total de atividades desse arquivo deve ser maior ou igual a 20 e deve contemplar todas as funcionalidades da aplicação.*

4 Python na Linha de Comando

Como dito antes, sua agenda de compromissos deve funcionar através da linha de comando. Para que isso funcione, Python deve estar instalada corretamente e acessível via terminal (para usuários de Linux e Mac OS) ou linha de comando (cmd.exe ou Powershell para usuários de Windows). Abra o terminal ou linha de comando e digite “python3” (sem as aspas). Se algo similar ao texto abaixo aparecer, então está tudo pronto para começar.

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Caso apareça uma mensagem de erro, você está enfrentando algum problema na configuração de Python. Converse o mais rápido possível com o professor ou algum monitor sobre isso. Para rodar um programa na linha de comando, é só digitar “python3” (novamente, sem as aspas) seguido do nome do arquivo “.py” que você quer executar. Por exemplo, para rodar o programa guardado no arquivo hipotético “prog.py”, é só digitar

```
python3 prog.py
```

na linha de comando. Adicionalmente, um programa Python pode receber informações via linha de comando quando o programa é executado. Por exemplo, se “prog.py” espera que o usuário forneça seu nome e sobrenome quando o programa for executado, isso pode ser feito da seguinte maneira:

```
python3 prog.py Fernando Castor
```

Uma vez que o programa seja executado dessa forma, essas informações (os nomes “Fernando” e “Castor”) ficam disponíveis para que o programa possa usá-las (ou ignorá-las). A próxima seção explica como.

5 Funcionamento da Agenda

Sua agenda deve ser implementada em um arquivo chamado "agenda.py". Ao final do projeto, sua agenda será capaz de **adicionar** uma nova atividade ao `todo.txt`, **listar** as atividades já registradas de maneira formatada, **remover** um compromisso previamente registrado, marcar um compromisso como **feito** e modificar a **prioridade** de um compromisso pré-existente. Para cada uma dessas funcionalidades, ela deverá ter um comando diferente, executável através da linha de comando. Por exemplo, para adicionar dois novos compromissos, um usuário poderia digitar o seguinte na linha de comando:

```
python3 agenda.py a 26102017 Comemorar meu aniversário @Casa
python3 agenda.py a Estudar programação todos os dias. @UFPE +IF968
```

O comando `a` precisa receber pelo menos a descrição da nova atividade, mas pode receber também todos os outros elementos (opcionais) explicados na Seção 3. Posteriormente, esse mesmo usuário poderia listar os compromissos já registrados usando o comando `l`:

```
python3 agenda.py l
```

O que imprimiria na tela o seguinte (supondo que o `todo.txt` não tinha outros compromissos):

```
1 26102017 Comemorar meu aniversário @Casa
2 Estudar programação todos os dias. @UFPE +IF968
```

Note os números 1 e 2 aparecendo antes das atividades. Esse número **não está no** `todo.txt` mas ele aparece quando o usuário lista as atividades. Esses números permitem, por exemplo, que o usuário remova um item ou modifique sua prioridade. O seu programa de agenda deve gerenciar a geração desses números sem que seja necessário armazená-los. Isso será explicado melhor posteriormente. Se o usuário quiser adicionar a prioridade "A" à segunda atividade, pode fazê-lo com o comando `p`:

```
python3 agenda p 2 A
```

Note que, para especificar a prioridade, o usuário não precisa colocar os parênteses. Depois disso, o conteúdo do arquivo `todo.txt` se tornará o seguinte:

```
1 26102017 Comemorar meu aniversário @Casa
2 (A) Estudar programação todos os dias. @UFPE +IF968
```

O usuário também pode marcar uma atividade como feita através do comando `f`, de "fazer":

```
python3 agenda.py f 2
```

O que esse comando faz é remover do `todo.txt` o compromisso selecionado e colocá-lo em um arquivo chamado `done.txt`. Por fim, usuários podem remover uma atividade usando o comando `r`:

```
python3 agenda.py r 1
```

6 Adicionando Novas Atividades

Abra o arquivo `agenda.py` e tente se familiarizar com o que o já está lá. Note que o arquivo tem várias funções cujas implementações estão vazias. Seu trabalho nesse projeto consistirá majoritariamente em implementar essas funções, embora você possa criar tantas funções novas quanto quiser. Para cada comando mencionado na seção anterior, você precisará implementar a lógica responsável por processá-lo a partir da linha de comando, assim como as funções que fazem o trabalho de fato quando um comando é recebido. Por exemplo, será necessário adicionar código à função `processarComandos()` para identificar que o comando `a` foi usado e que informações foram fornecidas mas também será necessário complementar a função `adicionar()` que já está no arquivo `agenda.py` para incluir essas informações no arquivo `todo.txt` conforme o esperado. Estes serão justamente seus primeiros trabalhos.

A função `processarComandos()` recebe como argumento uma lista de strings correspondendo às palavras passadas para Python quando o programa é executado. Dê uma olhada nos comentários no final do arquivo para entender como isso funciona. A implementação da função mostra como o comando `a`, fornecido a partir da linha de comando, é processado. Primeiro, a função verifica se o comando é de fato `a`. Se for, remove os dois primeiros elementos da lista `comandos` (o nome do arquivo `agenda.py` e o comando `a`). Tudo que sobra é informação relativa à nova atividade a ser registrada. Em seguida, é chamada a função `organizar`, que está vazia no momento. Essa função deve receber uma **lista** de strings com as informações a ser processadas, produzindo como resultado uma lista de tuplas no formato `(DESC, PRI, (DATA, HORA, CONTEXTO, PROJETO))`. Na função `processarComandos()`,

é criada uma lista cujo único elemento é o resultado de chamar `' '.join(comandos)`, porque estamos interessados apenas em processar as informações fornecidas pela linha de comando. Posteriormente, `organizar` será usada para processar todas as linhas do arquivo `todo.txt`.

Nosso primeiro trabalho mais direto no projeto é implementar a função `organizar`. Para cada linha de texto recebida, ela extrai os componentes da atividade (data, descrição, prioridade, etc.), valida cada um e constrói uma tupla contendo as informações devidamente organizadas. Para implementá-la, você deve realizar as tarefas a seguir:

Tarefa 3: *Complete a implementação da função `horaValida()` . Essa função recebe um string e verifica se ele tem exatamente quatro caracteres, se tão todos dígitos, se os dois primeiros formam um número entre 00 e 23 e se os dois últimos formam um número inteiro entre 00 e 59. Se tudo isso for verdade, ela devolve `True`. Caso contrário, `False`. O arquivo já inclui uma função auxiliar para verificar se todos os caracteres de um string são dígitos.*

Tarefa 4: *Implemente a função `dataValida()` . Essa função recebe um string e verifica se ele tem exatamente oito caracteres, se tão todos dígitos e se os dois primeiros correspondem a um dia válido, se o terceiro e o quarto correspondem a um mês válido e se os quatro últimos correspondem a um ano válido. Sua função deve checar também se o dia e o mês fazem sentido juntos. Além de verificar se o mês é um número entre 1 e 12, `dataValida()` deve checar se o dia poderia ocorrer naquele mês, por exemplo, ela deve devolver `False` caso o dia seja 31 mas o mês seja 04, que tem apenas 30 dias. O ano pode ser qualquer número de 4 dígitos. Para fevereiro, considere que pode haver até 29 dias, sem se preocupar se o ano é bissexto ou não. Se todas as verificações passarem, a função devolve `True`. Caso contrário, `False`.*

Tarefa 5: *Implemente a função `projetoValido()`. Essa função recebe um string e verifica se ele tem pelo menos dois caracteres e se o primeiro é `'+'`. Devolve `True` se as verificações passarem e `False` caso contrário.*

Tarefa 6: *Implemente a função `contextoValido()`. Essa função recebe um string e verifica se ele tem pelo menos dois caracteres e se o primeiro é `'@'`. Devolve `True` se as verificações passarem e `False` caso contrário.*

Tarefa 7: *Implemente a função `prioridadeValida()`. Essa função recebe um string e verifica se ele tem exatamente três caracteres, se o primeiro é `'('`, se o terceiro é `')'` e se o segundo é uma letra entre `A` e `Z`. A função deve funcionar tanto para letras minúsculas quanto maiúsculas. Devolve `True` se as verificações passarem e `False` caso contrário.*

Tarefa 8: *Complete a implementação da função `organizar()`. Como dito antes, essa função recebe uma lista de strings representando atividades e devolve uma lista de tuplas com as informações dessas atividades organizadas.*

Tarefa 9: *Complete a implementação da função `adicionar()`, que adiciona um compromisso à agenda. Um compromisso tem no mínimo uma descrição. Adicionalmente, pode ter, em caráter opcional, uma data, um horário, um contexto e um projeto. Esses itens opcionais são os elementos da tupla `extras`, o segundo parâmetro da função. Veja os comentários do código para saber como essa tupla é organizada. Todos os elementos dessa tupla precisam ser validados (com as funções definidas nas tarefas anteriores). Qualquer elemento da tupla que não passe pela validação deve ser ignorado.*

7 Listando Atividades

Agora que já é possível adicionar novas atividades à agenda, vamos estender o programa para que seja possível também listar as atividades já registradas, de maneira organizada e bonita. A função responsável por implementar essa funcionalidade chama-se `listar()`. Antes de ir para ela, porém, é necessário estender `processarComandos()` para que funcione também com o comando `l`. Esse comando não recebe informações adicionais. Posteriormente, uma tarefa opcional pedirá que você modifique esse comportamento.

Tarefa 10: *Modifique a função `processarComandos()` para que, ao receber o comando `l`, invoque a função `listar()`.*

A função `listar()` deve funcionar da seguinte maneira. Primeiro, ela **abre** o arquivo `todo.txt` e lê todo o seu conteúdo. Depois, **usa** `organizar()` para transformar as linhas de texto em tuplas organizadas, mais fáceis de processar. Terceiro, ela **ordena** a lista de tuplas com base nas prioridades das tarefas. As com prioridade `A` aparecem primeiro, depois as com prioridade `B`, etc. No final, aparecem as tarefas sem prioridade. Tarefas com o mesmo nível de prioridade (ou sem prioridade) devem ser ordenadas por data (mais cedo aparecendo primeiro) e em seguida por horário (mais cedo aparecendo primeiro). Tarefas sem prioridade, sem data e sem hora aparecem no final, sem ordem. Por fim, a função imprime os itens de maneira formatada. Atividades associadas com os 4 primeiros níveis de prioridade (`A`-`D`) devem ser coloridas de forma diferente (ver a função `printCores`), com o

nível `A` sendo colorido e também colocado em negrito. Cada item impresso deve ser precedido por um número. Esse número é gerado na hora da impressão e será usado posteriormente pelas operações de remoção, priorização e realização de atividades para identificá-las. Note que seu programa **não deve guardar esse número em lugar nenhum**. Por exemplo, se o arquivo `todo.txt` tiver o conteúdo apresentado a seguir

```
23052017 1030 Reunião com Huguinho, Luizinho e Zezinho. @Skype +Pesquisa
Comer menos açúcar
(B) Terminar a especificação do projeto de IF968. +IF968
```

o resultado do comando listar será algo como (sem as cores):

```
3 (B) Terminar a especificação do projeto de IF968. +IF968
1 23/05/2017 10h30m Reunião com Huguinho, Luizinho e Zezinho. @Skype +Pesquisa
2 Comer menos açúcar
```

Note que cada atividade tem um número e esse número não necessariamente reflete a ordem em que esses itens apareceram listados. Tais números podem ser atribuídos, por exemplo, com base na ordem em que os itens aparecem no arquivo. O importante é que essa atribuição de números seja consistente. Dessa forma, quando um usuário posteriormente remover uma atividade usando

```
python3 agenda.py r 2
```

ele removerá a atividade "Comer menos açúcar" e não "23052017 1030 Reunião...". Em outras palavras, há uma separação entre a forma como os dados são armazenados e a forma como são exibidos.

Tarefa 10: *Modifique a função `processarComandos()` para que, ao receber o comando `l`, invoque a função `listar()`.*

Tarefa 11: *Modifique a função `listar()` para ler o conteúdo do arquivo `todo.txt` em uma lista de strings e organizar esses strings em uma lista de tuplas, usando a função `organizar()`.*

Tarefa 12: *Construa uma função `ordenarPorDataHora()` que, dada uma lista de itens como a produzida por `organizar()`, com os itens já ordenados por prioridade, devolve uma lista que tem os mesmos itens, ordenados com base em suas datas e horas. Quanto mais antiga a data de um item, mais próximo do topo da lista o item deve estar. Itens que não têm data ou hora aparecem sempre no final, sem nenhuma ordem em particular. Modifique a função `listar()` que faça uso de `ordenarPorDataHora()`.*

Tarefa 13: *Construa uma função `ordenarPorPrioridade()` que, dada uma lista de itens como a produzida por `organizar()`, devolve uma lista que tem os mesmos itens, ordenados com base em suas prioridades, onde itens com prioridades mais altas (e.g., `A`), aparecem antes daqueles com prioridades mais baixas (e.g., `Z`). Itens que não têm prioridade aparecem sempre no final, sem qualquer ordem particular. Sua função deve garantir que, se uma lista de itens já estava ordenada por data e hora, essa ordem é mantida para cada prioridade (mas não entre prioridades). Por exemplo, se antes a lista estava ordenada por data e havia nela os seguintes itens:*

```
20052017 (B)
21052017 (A)
22052017 (B)
```

Após a execução de `ordenarPorPrioridade()`, a lista passaria a estar ordenada da seguinte maneira:

```
21052017 (A)
20052017 (B)
22052017 (B)
```

ou seja, o item com a prioridade `A` passou a aparecer primeiro mas os itens com prioridade `B` continuam apresentando a mesma ordem entre si. Modifique a função `listar()` que faça uso de `ordenarPorPrioridade()`.

Tarefa 14: *Modifique a função `listar()` para que liste as atividades no arquivo `todo.txt`. Os itens devem ser listados na ordem definida anteriormente, com itens nas prioridades `A-D` aparecendo em cores e itens com prioridade `A` também em negrito. Além disso, deve aparecer a numeração dos itens, como explicado antes nesta seção, de modo que possa ser usada pelas funções da próxima seção.*

Opcional 0: *Modifique seu programa para que o usuário também possa passar um critério de filtragem ao invocar o programa com o comando `l`. Esse critério de filtragem pode ser uma prioridade, um contexto ou um projeto e o comando `l` nestes casos deve listar apenas as atividades que têm essa mesma prioridade, contexto ou projeto.*

8 Removendo, Fazendo e Priorizando Atividades

Esta seção depende diretamente da anterior.

Tarefa 15: *Modifique a função* `processarComandos()` *para que, ao receber o comando* `r` *e o número de uma atividade, invoque a função* `remover()` *passando esse número como parâmetro.*

Tarefa 16: *Construa a função* `remover()` *que, dado o número de uma atividade, remove essa atividade do arquivo* `todo.txt`*. Se a atividade com esse número não existir, a função deve imprimir uma mensagem de erro.*

Tarefa 17: *Modifique a função* `processarComandos()` *para que, ao receber o comando* `p` *e o número de uma atividade, invoque a função* `priorizar()` *passando esse número como parâmetro.*

Tarefa 18: *Construa a função* `priorizar()` *que, dados o número N de uma atividade e uma prioridade P, modifica a prioridade dessa atividade N para que se torne P. Se essa atividade já tiver outra prioridade, ela é sobrescrita. Se a atividade não existir, a função deve imprimir uma mensagem de erro.*

Tarefa 19: *Modifique a função* `processarComandos()` *para que, ao receber o comando* `p` *e o número de uma atividade, invoque a função* `priorizar()` *passando esse número como parâmetro.*

Tarefa 20: *Construa a função* `priorizar()` *que, dados o número N de uma atividade e uma prioridade P, modifica a prioridade dessa atividade N para que se torne P. Se essa atividade já tiver outra prioridade, ela é sobrescrita. Se a atividade não existir, a função deve imprimir uma mensagem de erro.*

Tarefa 21: *Modifique a função* `processarComandos()` *para que, ao receber o comando* `f` *e o número de uma atividade, invoque a função* `fazer()` *passando esse número como parâmetro.*

Tarefa 22: *Construa a função* `fazer()` *que, dados o número N de uma atividade, marca essa atividade como feita. Isso significa que a atividade é removida do* `todo.txt` *e movida para o* `done.txt`*.*

9 Avaliação e Logística

O projeto contará como uma das três notas da disciplina e terá o mesmo valor que uma das provas. Não entregar o projeto implica imediatamente em nota 0 (ZERO) para os membros da equipe que não entregou. A avaliação se dará através de apresentação do projeto desenvolvido em sala de aula, para o professor ou algum dos monitores. Ausência no momento da apresentação do projeto implica em nota 0 (ZERO) para todos os membros faltosos da equipe.

A nota máxima que pode ser obtida no projeto é igual a 10 * N, onde N é o número de membros da equipe. Essa nota é atribuída pelo professor principalmente com base na apresentação e a distribuição dos pontos entre os membros da equipe (caso haja mais de um) é de responsabilidade dos mesmos. A nota máxima que cada membro da equipe pode obter é igual a 10.

Algumas considerações adicionais, relativas à logística do projeto:

- O trabalho poderá ser feito em grupos de no máximo dois componentes
- Os grupos devem se inscrever editando a página de equipes no sítio da disciplina, no endereço `https://sites.google.com/a/cin.ufpe.br/if968si/equipes` até **07/06**.
- O código completo do programa deverá estar disponível em um repositório no Github (`http://www.github.com`). Deve ser enviado um **email** para o professor com assunto **PROJETO IF968** até às **23h59 do dia 18/06/16** informando o endereço do repositório. Programas não entregues no horário serão penalizados com a perda de pontos, à taxa de 2,0 pontos perdidos para cada 24h de atraso.
- Trabalhos copiados de colegas ou da internet, seja trechos ou totalidade, serão prontamente anulados (todos os envolvidos).
- A apresentação dos projetos será no **dia 19/06 impreterivelmente**. Todos os membros devem saber todos os detalhes da implementação. O professor escolherá na hora a quem dirigirá cada pergunta durante a apresentação.