Projeto final da disciplina (nota AV3, Implementação)

Prezados alunos, chegou a hora do trabalho final da disciplina de Programação Funcional! Vamos às regras:

- **Cálculo da nota**:
  - **Definição de papéis (0,5 pts)**
    - O grupo deverá **obrigatoriamente** formalizar um papel para cada um dos integrantes. Uma pessoa (ex: Astrogildo) pode, por exemplo, ficar responsável pela documentação dos requisitos, enquanto outras duas pessoas (ex: Creuzilene e Asclepíades) ficam encarregadas do código, e outra pessoa fica encarregada dos testes. No entanto, mesmo com os papéis rigorosamente identificados, nada impede que, por exemplo, alguém da parte de implementação ajude também na parte de documentação, e assim por diante… Caso o grupo esqueça de formalizar uma tarefa para algum integrante, o grupo ficará sem a pontuação referente à definição de papéis;
  - **Uso dos conceitos estudados na disciplina de Programação Funcional (3,5 pts)**:
    - O grupo deverá utilizar os seguintes conceitos:
      - Ao menos uma função lambda de alta ordem: 0,5 pts;
      - Ao menos uma função lambda recursiva: 0,5 pts;
      - Ao menos uma ocorrência de sucessivas chamadas de funções lambda utilizando currying: 0,5 pts;
      - Ao menos uma List Comprehension dentro do escopo de uma Lambda: 0,5 pts;
      - Ao menos um dicionário dentro do escopo (no parâmetro, ou no retorno) de uma função lambda: 0,5 pts;

        - Ao menos um entre os seguintes functores: map, filter, reduce: 0,5 pts;
        - Ao menos um monad construído pelo grupo: 0,5 pts (fiquem atentos a como construir um monad… se tiver sido feito errado, a pontuação referente a este conceito poderá ser negada… **não vale utilizar try-catch ou similares**… **Se o monad for construído corretamente, ele também poderá contar como uma função de alta ordem (e assim satisfazendo o outro requisito da questão)**);
    - **OBS: indiquem, no relatório de vocês, onde estão presentes estas funções**!
    - Uma vez utilizados os conceitos acima ao menos uma vez, o grupo não precisará ficar "amarrado" apenas à Programação Funcional (diferentemente das regras exigidas para os exercícios da AV2);
- **Requisitos (diferenciando funcionais e não-funcionais) e sua fidedignidade com o código (2,0 pts)**
    - Vocês podem dedicar uma seção sobre os requisitos no relatório do trabalho de vocês;
    - Tenham bom senso na escolha dos requisitos. Busquem um conjunto de requisitos que não seja simples demais, ou abrangente/complexo demais a ponto de tornar o trabalho infactível para o prazo de 1 mês e 10 dias. Busque o equilíbrio…;
    - A fidedignidade do código com os requisitos será medida checando se todas as funções do sistema estão documentadas no relatório, e se todas as funções citadas pelo documento estão presentes no código;
    - **IMPORTANTE: não esqueça de citar onde cada requisito está implementado no código**! Por exemplo, vocês deverão escrever algo como:

        "O requisito *[RNF 001]*, que indica que o sistema deve ter Confidencialidade, está implementado

na função `check_credentials (login, senha)`, presente no módulo `Conf.py`”

- **Testes executados pelo professor (2 pts)**:
  - Esta tarefa servirá como termômetro para medir a qualidade do sistema implementado. O professor criará 5 casos de testes para cada grupo, e, durante o período reservado para as apresentações do trabalho, pedirá ao grupo para executá-los… Cada caso de teste valerá 0,4 pontos;
- **Progresso entre sprints (2 pts)**: em cada semana, o professor organizará sprints com os grupos, para apurar o andamento do trabalho, e se houve progresso de uma quinzena para a outra. **Cada sprint deverá fornecer evidências de progresso do trabalho entre uma semana e outra, possivelmente solucionando pendências de backlog**. Cada sprint que tiver registrado progresso valerá 0,5 ponto (já que são 4 sprints, o total pode alcançar 2,0);

- **Acompanhamento do trabalho**:
  - O professor organizará sprints semanais para monitorar e acompanhar o desenvolvimento do projeto para cada grupo. Como dito no tópico anterior, as sprints deverão fornecer avanços em relação à sprint anterior. A cada interação com o professor, novos requisitos ou problemas podem surgir, e eles deverão ser incluídos no trabalho (no documento e na implementação);
- **Disponibilização do trabalho**:
  - O código-fonte e o relatório do trabalho deverão ser disponibilizados no *github* (GitHub: Let’s build from here · GitHub), em uma conta pertencente a algum integrante do grupo, ou em uma conta criada pelo próprio grupo;
- **Sobre a avaliação do trabalho**:
  - **Os trabalhos serão avaliados estritamente com base nos critérios estabelecidos aqui**. Fiquem tranquilos que não haverá comparação entre trabalhos. Por exemplo: suponha que o grupo do Pacman fez um trabalho vistosamente maravilhoso, com uma mega-interface gráfica tridimensional, trilha sonora, e em que o personagem exibe expressões

faciais e etc, mas não redigiu os requisitos no relatório… Este grupo ficará sem 2 pontos automaticamente, independente do quão impactante tenha sido a implementação. Já um outro grupo, suponha, que fez o Sokobox com interface simples e satisfez tudo o que foi exigido nas regras, com o sistema passando por todos os casos de testes… Este grupo poderá ficar com 10,0;
    - O grupo **não poderá reaproveitar trabalhos** de colegas de outros semestres, podendo sofrer penalizações na nota final;
    - O grupo poderá utilizar o auxílio de chatbots (como o ChatGPT), mas deverá documentar nos requisitos como utilizou o chatbot (quais foram as perguntas feitas a ele);
    - Cada **grupo terá uma nota**, e esta **será a nota de TODOS os integrantes do grupo** na AV3;

Agora, vamos às sugestões de temas para o trabalho de vocês…

Compilador e Tradutor de código (grupos com linguagens de entrada diferentes)

Este projeto visa a implementação de um interpretador de um código-fonte (o código de entrada do programa) em alguma linguagem de programação. Esta interpretação pode ser alcançada traduzindo-se o programa escrito na linguagem-fonte para a linguagem alvo (por exemplo, Python). Por exemplo, se o seu programa receber como entrada um subconjunto da linguagem *QBASIC*, o seu programa pode pegar um arquivo de texto com o seguinte código:

```
PRINT "Guess a number between 1 and 10: ";
INPUT num
IF (num < 1 OR num > 10) THEN
     PRINT "That is not between 1 and 10"
END IF
IF (num = 6) THEN
     PRINT "Correct!!!"
ELSE
     PRINT "Try again"
```

```
END IF
```

e retornar um código executável em Python (a linguagem alvo) com o seguinte código:

```
print ("Guess a number between 1 and 10: ");
num = int (input())
if num < 1 or num > 10 :
    print ("That is not between 1 and 10")
if num == 6 :
    print ("Correct!!!")
else :
    print ("Try again")
```

Como a intenção é interpretar o código-fonte dado como entrada, o programa em Python gerado deverá ser executado após a conversão.

Dicas úteis para o desenvolvimento deste projeto

Como as entradas serão longas (já que correspondem a um código em um programa), recomendo que sejam armazenadas em arquivos de texto (com o formato .txt, ou mesmo com o formato correspondente à linguagem de entrada), e capturadas através das seguintes instruções:

```
f = open ("text_file.txt")
print (f.read())
```

A instrução `f.read()` é uma string, e um provável passo seguinte a este é a tokenização desta string. Tokenizar a string significa dividi-la em várias strings menores utilizando algum caractere separador (no nosso caso, o espaço em branco e os parênteses). Logo, após tokenizada, a string `strif` a seguir

```
“IF ( num < 1 OR num > 10 ) THEN PRINT \"That is not between 1 and 10\” END IF”
```

será convertida na seguinte lista de strings:

```
[“IF”, “(“, “num”, “<”, “1”, “OR”, “num”, “>”, “10”, “)”,
“THEN”, “PRINT”, “\"”, “That“, “is“, “not“, “between“,
“1“, “and“, “10“, “\””, “END”, “IF”]
```

A tokenização pode ser feita em Python utilizando-se o seguinte comando:

```
list_of_tokens = strif.split()
```

Agora vem a parte interessante do trabalho… Como será a lógica para converter um bloco de código dado na linguagem de entrada (um *if-statement* em *QBASIC*, por exemplo) em outro bloco de código na linguagem de saída? Vamos refletir para o caso do `IF`: um `IF`-statement sempre terá que começar com a palavra reservada `IF`, e deverá ser seguida por uma expressão booleana, depois pela palavra `THEN`, depois por outro bloco de código, e depois pelas palavras reservadas `END` e `IF`. logo, a função que traduz um bloco de código `IF` em *QBASIC* para um bloco de código `if` em Python pode ser dada como segue:

```
convert_if =
     lambda ifcode :
          rec_gen_word (“IF”) + rec_gen_expression (sublist1)
          + rec_gen_word (“THEN”) + rec_gen_block (sublist2)
          + rec_gen_word (“END”) + rec_gen_word (“IF”)
```

A função `rec_gen_word` pode ser encarregada de reconhecer e traduzir para a linguagem-alvo (código-objeto) todas as palavras reservadas da linguagem dada como entrada. Por exemplo, se você estiver implementando um compilador de *QBASIC* para *Python*:

- `rec_gen_word (“IF”)` pode retornar `“if”`;
- `rec_gen_word (“VIXE”)` retornaria erro de compilação;

Para o reconhecimento de tokens relacionados a nomes de variáveis (identificadores) e expressões numéricas, você pode utilizar `RegEx`, que vimos em sala de aula.

Um aspecto importante do trabalho é que você faça o reconhecimento e tradução das expressões booleanas e aritméticas de forma adequada. Não caia na armadilha de fazer um reconhecimento sequencial simples, pois o seu programa também deve ser capaz de reconhecer e vetar expressões que levam a erros sintáticos.

Para facilitar o reconhecimento dos tokens, você poderá:

- exigir ao usuário do programa que seja digitado um espaço ou um enter sempre que um parêntese/colchete/chave seja digitado;
- O subconjunto de entrada da linguagem-fonte será restrito aos blocos de código relacionados a
    - declaração, implementação e chamada de funções;
    - *if-then-elses* e *for/while*;
    - entradas e saídas pelo terminal;
    - instruções de atribuição;
    - expressões numéricas e booleanas;

Sistema Bancário, de Loja Virtual, ou qualquer outro que possa envolver Integração com Banco de Dados

Esta proposta de trabalho visa a implementação de um sistema que envolve a Integração com algum Banco de Dados. Assim, estes sistemas requerem propriedades inerentes a Bancos de Dados. Um banco de dados, seja ele qual for (e com este trabalho não é diferente), tem algumas propriedades muito importantes que devem ser satisfeitas:

- Integridade: os dados armazenados devem ser íntegros e consistentes. Já pensou se você acessasse dados sobre um usuário que possui uma idade menor do que 0? Ou se acessasse um número de CPF que está associado a duas pessoas? Você não conseguiria confiar na integridade das suas informações, portanto você precisará tratar estas situações;
- Persistência: os dados armazenados são capazes de persistir (ou seja, de continuarem existindo mesmo que a máquina física que armazena os dados seja desligada)? No nosso caso, podemos considerar que esta propriedade é assegurada no momento em que armazenamos os nossos dados em arquivos-texto;

- Segurança: as informações estão seguras? Se elas são armazenadas em um arquivo-texto, será que alguém que tem acesso a este arquivo-texto conseguirá ler as informações ali armazenadas? O uso de criptografia na hora de armazenar os dados é crucial para garantir a segurança das informações do banco;
- Controle de acesso: para usuários comuns, apenas as suas informações devem poder ser consultadas. Apenas o administrador do banco de dados terá acesso a todas as informações;

Um banco de dados também possui operações (CRUD) que se esperam que sejam feitas nos dados:

- Criação (Create);
- Leitura (Read);
- Atualização (Update);
- Remoção (Delete);

Portanto, todas as propriedades acima deverão ser garantidas pelo seu projeto.

Dicas úteis para o desenvolvimento deste projeto

Você pode armazenar os dados seguindo a lógica de um **Banco de Dados chave-valor**. Neste caso, você poderá utilizar a estrutura de dados dicionário, que nós vimos nas aulas de Programação Funcional. Pode ser utilizada uma variável-dicionário similar àquelas que vimos nas aulas, mas este projeto terá que ser mais sofisticado do que aqueles exemplos que vimos. A persistência dos dados poderá ser alcançada armazenando-se os dados em um arquivo de texto. A segurança poderá ser alcançada criptografando-se as senhas dos usuários (você pode fazer uso da API *Crypto*, do Python, que nós vimos em sala de aula no exemplo sobre criptografia). A verificação de integridade pode ser feita estabelecendo e verificando-se condições de consistência do banco de dados.

Você também pode armazenar os dados seguindo a lógica de um **Banco de Dados Orientado a Grafos**. Neste caso, você pode implementar uma estrutura de dados Grafo no seu sistema utilizando tuplas e dicionários. Um grafo pode ser visto como um conjunto de transições entre um nó/node/vértice inicial e um nó/node/vértice final através de uma aresta/edge. Em um Banco de Dados Grafo, arestas/edges são os relacionamentos (relationships). Cada transição pode ser armazenada como uma tupla com 3 valores:

```
(source_node, edge_relationship, destination_node)
```

Outra forma interessante de organizar os seus dados é através de dicionários cujos nomes são os relacionamentos, cujas chaves são os nós-fonte, e cujos valores são os nós-destino. Ex:

```
orientacao = {
    "Allan Turing" : "Alonzo Church"
    , "Edsger Dijkstra" : "Adriaan van Wijngaarden"
    , "Donald Knuth" : "Marshall Hall"
}
```

Não é obrigatório, mas você pode utilizar integrações de Python com outros bancos de dados (e.g. *Neo4J*, *MySQL* ou *Dynamo-DB*), Ok? Entretanto, mesmo que você opte por não utilizar, não se assuste com o que está sendo pedido: todo o conhecimento que você acumulou nesta disciplina te dará condições de desenvolver o projeto… Fique tranquilo!

Jogo 2D (Pacman, Sokobox, Cobrinha, Labirinto ou etc)

Este projeto consiste em implementar um jogo 2D com interface simples (não precisa ser Interface Gráfica com o Usuário sofisticada, como por exemplo *tkinter*… você pode capturar pressão de teclas e imprimir o cenário em tela de console mesmo). Este jogo deve ser extensível para qualquer número de fases, e o cenário de cada fase pode ser especificado em um arquivo de texto que será lido pelo seu programa.

Dicas úteis para o desenvolvimento deste projeto

Para capturar o pressionar de uma tecla, você pode fazer uso da API `keyboard` do Python, aliada à API `time`, e utilizar o seguinte código como fonte de inspiração (**que tal você convertê-lo para Lambda como exercício?**):

```
import keyboard
import time
count = 0
time_pressed = 0.3
just_wait = lambda : print ("", end = "")
while True:
    try:
        if keyboard.is_pressed('>'):
            print('Pacman, move to the right!')
            time.sleep (time_pressed)
        elif keyboard.is_pressed('<'):
            print('Pacman, move to the left!')
            time.sleep (time_pressed)
    except:
        just_wait ()
```

Algumas observações (podem existir outras também… e estas podem fazer parte dos requisitos):

- É interessante que os cenários (fases) sejam armazenados em arquivos de texto (strings) que serão lidos na inicialização do seu programa/jogo. Desta forma, você pode, de forma simples, estender o seu jogo para várias fases diferentes;
- Você deve ficar atento aos obstáculos das fases para que o personagem não os ultrapasse, mesmo após pressionar uma tecla direcional;
- Os movimentos dos fantasmas (no caso do Pacman) podem ser implementados seguindo uma lógica randômica…
- No caso do Sokobox, se o personagem estiver adjacente a alguma caixa e ele se mover do lado oposto, a caixa também deverá se mover, caso não haja uma parede depois;

Calculadora de Integrais ou Derivadas

(OBS: NÃO será permitido utilizar APIs de terceiros aqui como, por exemplo, SymPy!)

Este projeto pode pegar como entrada um comando que recebe, entre outras entradas, uma expressão, e calcula a integral daquela expressão. Um comando possível poderia ser:

```
integral <expression> d <variable_name>
```

Este projeto é interessante porque o cálculo de integrais não é tão trivial quanto inicialmente pode-se pensar… existem diferentes métodos de integração que podem ser utilizados, dependendo de como estiver a expressão de entrada… Seguem alguns métodos de integração:

- Integração por substituição
- Integração por partes
- Integração por frações parciais