Especificação do Trabalho 1

Objetivos

1. Usar os conhecimentos abordados para o projeto de funções.
2. Escreva corpos de função usando variáveis, tipos numéricos, strings e instruções condicionais.

O jogo

Neste trabalho você ajudará a escrever um jogo de quebra-cabeças de frases.

Há um arquivo, apenas, que precisa da sua contribuição: auxiliares.py. Enquanto você não implementar algumas funções lá, não será capaz de jogar.

Há três formas de execução do jogo que são mostradas em um menu quando você executa o projeto a partir de jogo.py:

```
Escolha um tipo de jogo:
  [H-] - Um jogador
  [HH] - HUMANO-HUMANO
  [HC] - HUMANO-COMPUTADOR
```

Por exemplo, se selecionado o jogo de um jogador:

```
H-
```

Aparecerá algo como:

```
jogador Um, é sua vez. Você tem 0 pontos.


^ ^ ' ^  ^ ^ ^ ^  ^ ^ ^ ^ ^ ^ ^ ^ ^

Selecione o tipo de movimento:
[V] - VOGAL,
[C] - CONSOANTE,
[R] - RESOLVER,
[S] - SAIR.
```

^ indica uma letra escondida na frase, que você terá que adivinhar. Outros caracteres como espaços em branco e apóstrofos serão exibidos. Suponha que você escolheu consoante.

C

Aparecerá para você algo como:

Escolha uma letra de [b,c,d,f,g,h,j,l,m,n,p,q,r,s,t,v,w,x,y,z]: m

Jogador Um palpita m, que ocorre 1 vez(es) no quebra cabeça.

Pontuação de Jogador Um é 1.

==========

jogador 1, é sua vez. Você tem 1 pontos.

^ ^ ' ^ ^ ^ m ^ ^ ^ ^ ^ ^ ^ ^ ^ ^

Selecione o tipo de movimento:

[V] - VOGAL,

[C] - CONSOANTE,

[R] - RESOLVER,

[S] - SAIR.

Se você repetir o procedimento com uma escolha de letra errada, ocorre algo como:

C

Escolha uma letra de [b,c,d,f,g,h,j,l,m,n,p,q,r,s,t,v,w,x,y,z]: z

Jogador Um palpita z, que ocorre 0 vez(es) no quebra cabeça.

Pontuação de Jogador Um é 1.

==========

jogador Um, é sua vez. Você tem 1 pontos.

^ ^ ' ^ ^ ^ m ^ ^ ^ ^ ^ ^ ^ ^ ^ ^

Selecione o tipo de movimento:

[V] - VOGAL,

[C] - CONSOANTE,

[R] - RESOLVER,

[S] - SAIR.

Ou seja, o jogo permanece como estava anteriormente. Caso queria adivinhar vogais a sua interação com o jogo será semelhante a:

V

Escolha uma letra de [a,e,i,o,u]: a

Jogador Um palpita a, que ocorre 2 vez(es) no quebra cabeça.

Pontuação de Jogador Um é 0.

==========

jogador Um, é sua vez. Você tem 0 pontos.

^ ^ ' ^ ^ ^ m ^ ^ ^ ^ a ^ ^ a ^ ^

Selecione o tipo de movimento:

[V] - VOGAL,

[C] - CONSOANTE,

[R] - RESOLVER,

[S] - SAIR.

A pontuação em vogais diminui, porque você compra o direito de adivinhar vogais. Assim, é um custo, cada vogal custa 1 ponto. No caso de tentar adivinhar novamente, você obterá a seguinte saída:

V

Você não tem pontos suficientes para revelar uma vogal. Vogais custam 1 ponto(s).

Selecione o tipo de movimento:

[V] - VOGAL,

[C] - CONSOANTE,

[R] - RESOLVER,

[S] - SAIR.

Nessa situação, você terá que adivinhar consoantes para poder ganhar pontos que permitirá a adivinhação de vogais.

À medida que forem adivinhando vogais e consoantes, a frase escondida por ^ vai sendo revelada, e você pode chegar na situação onde será capaz de tentar adivinhar toda a frase de uma única vez:

R

Informe seu palpite: it's time for breakfast

E o vencedor é... Jogador Um!

A solução para o quebra cabeça deste jogo é: it's time for breakfast.

Neste jogo, o jogador marcou 17 ponto(s)

A pontuação 17, no caso do exemplo, se deve a dois pontos para cada consoante que estava escondida no momento da adivinhação mais a pontuação anterior. No caso do exemplo, havia 3 pontos para o jogador e quando ele adivinhou a frase, havia 7 consoantes escondidas, que dá 14 pontos. Assim, 3+14 = 17.

E neste ponto, o programa encerra.

Código inicial

Para este trabalho, alguns arquivos são fornecidos, incluindo arquivos de código inicial do Python. Extraia o arquivo zip.

Existem três arquivos Python no código inicial, dois arquivos de entrada de texto de amostra:

- auxiliar.py: Este arquivo contém algum código que configura o uso de constantes de constantes.py, uma função auxiliar para você usar (consulte a tabela de funções a serem implementadas abaixo), bem como o cabeçalho e a docstring completa (mas não o corpo) para a primeira função que você deve escrever. Seu trabalho é completar este arquivo.
- constantes.py: Este arquivo contém algumas constantes úteis que você deve usar em sua solução. Você não modificará este arquivo.
- jogo.py: Este é o programa principal. Você não modificará este arquivo. Em vez disso, você executará este arquivo para jogar o jogo. Observe que jogo.py depende das funções que você escreverá em auxiliares.py, portanto, este programa não será executado corretamente até que você implemente as funções.
- jogo-pequeno.txt e jogo-grande.txt Cada um desses arquivos contém frases de quebra-cabeça usadas por jogo.py para selecionar um quebra-cabeça para um jogo e também para fornecer um conjunto de frases que o jogador do computador "conhece". Você pode usar jogo-pequeno.txt quando estiver no início do desenvolvimento e depuração de sua solução e jogo-grande.txt para jogar o jogo.

Constantes

Constantes são variáveis especiais cujos valores não devem mudar depois de atribuídos. Uma convenção de nomenclatura diferente (sempre em maiúsculas) é usada para constantes, para que os programadores saibam que não devem alterar seus valores. Por exemplo, no código inicial, a constante PONTOS_CONSOANTE recebe o valor 1 no início do módulo e o valor de PONTOS_CONSOANTE nunca deve mudar em seu código. Ao escrever seu código, se você precisar usar o número de pontos atribuídos para uma consoante adivinhada, você deve usar PONTOS_CONSOANTE. O mesmo vale para os outros valores constantes.

O uso de constantes simplifica as modificações de código e melhora a legibilidade e a mutabilidade. Por exemplo, se mais tarde decidirmos usar um caractere diferente para representar um índice oculto em nosso quebra-cabeça, teríamos apenas que fazer uma mudança em um lugar (a instrução de atribuição ESCONDIDA), em vez de em todo o programa. Isso também torna nosso código mais legível - quer usemos '^' ou qualquer outro caractere para representar um índice oculto, escrevemos nosso código usando a constante ESCONDIDApara que fique claro para o leitor o que queremos dizer.

Mais sobre o programa

- Duas strings são usadas para representar informações sobre um quebra-cabeça de frase:
  - O quebra-cabeça é composto de caracteres alfabéticos e não alfabéticos (por exemplo, espaços, pontuação e dígitos). Um exemplo de string de quebra-cabeça é ‘e-mail inbox'.
  - visão é a visão atual do quebra-cabeça vista pelos jogadores. Nele, os caracteres alfabéticos são exibidos (se foram revelados) ou escondidos (usando um caractere especial (um acento circunflexo '^' por padrão), se ainda não foram revelados). Caracteres não alfabéticos (espaços, pontuação e dígitos) são sempre exibidos. Por exemplo, no início do jogo, a string de visualização para o quebra-cabeça acima seria (com um acento circunflexo para representar um personagem oculto): '^ - ^^^^ ^^^^^'.

    - Conforme o jogo avança e os jogadores adivinham as letras a serem reveladas, a visualização é atualizada. Continuando com o exemplo acima, se o jogador adivinhar "m", a visualização torna-se '^ -m ^^^ ^^^^^' e, se "i" for adivinhado, torna-se '^ -m ^ i ^ i ^ ^^^ '.
- Existem três tipos de jogos que o seu programa irá jogar: humano, humano vs humano e humano vs computador. Usaremos as constantes HUMANO, HUMANO_HUMANO e HUMANO_COMPUTADOR respectivamente, para representar o tipo de jogo que está sendo jogado.
- Quando um jogador adivinha uma consoante, cada ocorrência daquela consoante no quebra-cabeça ganha um certo número de pontos para aquele jogador: o valor da constante PONTOS_CONSOANTE. Se um jogador adivinhar uma consoante corretamente, ele joga novamente no próximo turno. Se a estimativa estiver incorreta, a vez vai para o próximo jogador (em um jogo para dois jogadores).
- Os jogadores têm que pagar para adivinhar uma vogal. O custo não depende do número de ocorrências da vogal, e é sempre igual ao valor da constante PRECO_VOGAL. Assim, adivinhar uma vogal diminui os pontos do jogador. Um jogador não pode adivinhar uma vogal se não tiver pontos suficientes para pagar por isso. Depois de comprar uma vogal, se um jogador selecionar corretamente uma vogal no quebra-cabeça, ele joga novamente no próximo turno. Se a seleção da vogal estiver incorreta, a vez vai para o próximo jogador (em um jogo para dois jogadores).
- Quando um jogador resolve o quebra-cabeça, os pontos de bônus de BONUS_CONSOANTE são adicionados à sua pontuação para cada ocorrência de uma consoante que ainda está ESCONDIDA.
- Todos os quebra-cabeças são sempre minúsculos. Ou seja, os quebra-cabeças não conterão nenhuma letra maiúscula.
- Você deve ler o arquivo constantes.py cuidadosamente para entender o propósito de cada constante definida. Você deve usar essas constantes em seu código e NÃO os valores literais. Por exemplo, você deve usar ESCONDIDA em vez de '^'.

O que fazer

No arquivo de código inicial auxiliares.py, complete as seguintes definições de função. Use a Receita de Projeto de Função que você aprendeu neste curso. Incluímos os contratos de tipo na tabela a seguir; leia a tabela para entender como as funções serão usadas.

Nome da função: (Parâmetros) -> Retorno	Descrição
e_vitoria: (str, str): bool	O primeiro parâmetro representa a frase do quebra-cabeça. O segundo parâmetro representa a visualização atual. A função deve retornar True se e somente se eles representarem uma combinação vencedora, ou seja, eles são iguais.
e_fim_de_jogo: (str,str,str) -> bool	O primeiro parâmetro representa a frase do quebra-cabeça. O segundo parâmetro representa a visualização atual. O terceiro parâmetro representa o próximo movimento. A função deve retornar True se e somente se esta for uma combinação término do jogo, ou seja, ou esta é uma combinação vencedora ou o movimento é "sair".

e_jogo_de_um_jogador: (str) -> bool	O parâmetro representa o tipo de jogo que está sendo jogado: humano, humano-humano ou humano-computador. A função deve retornar True se e somente se este for um jogo para um jogador.
e_humano: (str, str) -> bool	O primeiro parâmetro representa o jogador atual - o jogador da vez - jogador um ou jogador dois. O segundo parâmetro representa o tipo de jogo que está sendo jogado: humano, humano-humano ou humano-computador. A função deve retornar True se e somente se o jogador atual for um humano.
pontuacao_jogador_atual: (int, int, str) -> int	O primeiro parâmetro representa a pontuação do jogador um. O segundo parâmetro representa a pontuação do jogador dois. O terceiro parâmetro representa o jogador atual - o jogador da vez. A função deve retornar a pontuação do jogador atual.
e_letra_bonus: (str, str, str) -> bool	O primeiro parâmetro representa uma letra. O segundo parâmetro representa o quebra-cabeça da frase. O terceiro parâmetro representa a visualização atual. A função deve retornar True se e somente se o primeiro argumento for uma letra bônus. Lembre-se de que as letras bônus são consoantes que estão ocultas no momento.
atualizar_visao_caractere: (str, str, int, str) -> str	O primeiro parâmetro representa o quebra-cabeça da frase. O segundo parâmetro representa a visualização atual. O terceiro parâmetro representa o índice de um caractere cuja visão estamos atualizando. O quarto parâmetro representa o palpite para esse caractere. A função retorna o que a visão atualizada do caracteredeve ser: se o palpite estiver correto, o caractere deve ser revelado; caso contrário, a visualização não deve ser alterada.
calcula_pontuacao: (int, int, str) -> int	O primeiro parâmetro representa a pontuação atual. O segundo parâmetro representa o número de ocorrências reveladas como resultado do movimento atual do jogador. O terceiro parâmetro representa o movimento atual: adivinhe uma consoante ou compre uma vogal. A função deve retornar a nova pontuação atualizada. Lembre-se de que adivinhar uma consoante aumenta a pontuação e o aumento depende do número de letras adivinhadas, e que comprar uma vogal diminui a pontuação em um valor fixo.
proximo_jogador: (str, int, str) -> str	O primeiro parâmetro representa o jogador atual - o jogador da vez -

	jogador um ou jogador dois. O segundo parâmetro representa o número de ocorrências reveladas pelo jogador atual como resultado do último movimento do jogador. O terceiro parâmetro representa o tipo de jogo que está sendo jogado: humano, humano-humano ou humano-computador. A função deve fazer com que o jogador volte a jogar na próxima jogada - jogador um ou jogador dois. Lembre-se de que se o jogador atual adivinhar corretamente uma consoante ou comprar uma vogal, o jogador joga novamente. Se o jogador adivinhar incorretamente, o outro jogador joga na próxima jogada. Claro, em um jogo de um jogador, o jogador não muda.
e_escondida: (int, str, str) -> bool	O primeiro parâmetro representa o índice de um caractere. O segundo parâmetro representa o quebra-cabeça da frase. O terceiro parâmetro representa a visualização atual. A função deve retornar True se e somente se o caractere no índice fornecido estiver oculto no jogo.
computador_escolhe_resolver: (str, str, str) -> bool	Esta função é usada para decidir o que fazer quando for a vez do computador jogar. O primeiro parâmetro representa a visualização atual. O segundo parâmetro representa a dificuldade do jogo atual: fácil ou difícil. O terceiro parâmetro representa as consoantes que ainda não foram adivinhadas neste jogo. A função deve retornar True se e somente se o computador decidir resolver o quebra-cabeça. Você pode presumir que essa função só é chamada em um jogo de computador humano. A estratégia que usaremos para o computador é a seguinte: Se a dificuldade for difícil, o computador decide resolver se pelo menos metade das letras foram reveladas ou se não há mais consoantes para adivinhar. Caso contrário, o computador não resolverá. Fornecemos uma função auxiliar esta_metade_revelada que você pode usar nesta função.
apague: (str, int) -> str	O primeiro parâmetro representa uma sequência de letras. O segundo parâmetro representa um índice. A função deve retornar as letras fornecidas com o caractere no índice fornecido removido, se o índice for um índice válido para aquela sequência de

	letras. Caso contrário, ela deve retornar a seqüência original de letras inalterada.

Usando constantes

Conforme discutimos na seção Constantes acima, seu código deve fazer uso das constantes fornecidas. Se o valor de uma dessas constantes for alterado e seu programa for executado novamente, suas funções deverão funcionar com esses novos valores.

Por exemplo, se PONTOS_CONSOANTE foi alterada, então suas funções devem funcionar de acordo com o novo número de pontos de bônus que devem ser ganhos por adivinhar uma consoante. Usar constantes em seu código significa que isso acontece automaticamente.

Seus exemplos de docstring devem refletir os valores fornecidos das constantes no código inicial fornecido e não precisam ser alterados.

Sem entrada ou saída

Auxiliares.py não deve incluir nenhuma chamada para as funções de saída e entrada. Não adicione nenhuma instrução de importação. Além disso, não inclua nenhuma chamada de função ou outro código fora das definições de função.

Como testar se código funciona

Certifique-se de seguir a receita de projeto de função ensinada na aula, você deve escrever 3-5 doctests por funções, para que possa testar seu código.

Nota

Estes são os aspectos do seu trabalho que podem ser avalaidos para nota:

- Estilo do código (20%):
  - Certifique-se de seguir as diretrizes de estilo do Python que apresentamos e as convenções de codificação do Python que temos usado ao longo do semestre.
  - Todas as funções, incluindo funções auxiliares, devem ter docstrings completas, incluindo pré-condições, quando você achar que elas são necessárias.
- Corretude (80%):
  - Suas funções devem ser executadas conforme especificado.

Como submeter

A submissão é feita no sigaa. Está liberada a ressubmissão.

Construa um arquivo compactado, tem que ser .zip, não serão aceitos outros tipos de compactação. Por exemplo, o aluno de matrícula 000000, vai gerar o arquivo 000000_Trabalho_1.zip contendo o seu código de auxiliares.py.