Pesquisa, ordenação, expressões lambda.

Preparação (antes da aula)

- Execute as seguintes instruções em modo interativo e interprete os resultados.

```
L = ["Mario", "Carla",
     "anabela", "Maria", "nuno"]
sorted(L)
sorted(L, reverse=True)
sorted(L, key=len)
L[0].casefold()
str.casefold(L[0]) #equivalent
sorted(L, key=str.casefold)
def lenFold(s):
  return (len(s), s.casefold())

lenFold(L[0])
sorted(L, key=lenFold)
```

```
D = [('Rui', 1998, 10, 5),
     ('Carla', 2000, 12, 25),
     ('Rita', 1998, 4, 25),
     ('Lia', 2001, 12, 1)]
sorted(D)
sorted(D, key=lambda t:(t[2],t[3]))

N = [3,4,4,4,6,7,7,8]
import bisect
bisect.bisect_left(N, 6)
bisect.bisect_left(N, 10)
bisect.bisect_left(N, 4)
bisect.bisect_right(N, 4)
```

Exercícios

1. Na aula07 fez um programa que conta ocorrências de letras num ficheiro de texto. Faça uma nova versão desse programa que liste o resultado por ordem decrescente do número de ocorrências. Use o método `sorted` com os argumentos `key=` e `reverse=` para ordenar a sequência de pares chave-valor (`items`) do dicionário.

```
$ python3 countLetters2.py wordlist.txt
e 61167
s 47192
i 45719
```

2. O programa `tabelaFutebol.py` tem uma lista com a tabela classificativa de um campeonato de futebol. Cada elemento da lista é um tuplo com o nome da equipa e números de vitórias, de empates, de derrotas, de golos marcados e sofridos. O programa já tem uma função para mostrar a tabela devidamente formatada e uma função, definida por uma expressão lambda, para determinar o número de jogos realizados por uma equipa. Complete o programa nos locais indicados para resolver cada uma das alíneas.

   a) Complete a expressão lambda para definir a função `pontos` que, dado um registo de uma equipa, deve devolver o número de pontos da equipa. (Cada vitória vale 3 pontos, cada empate vale 1 ponto.)

   b) Acrescente os argumentos adequados à função `sorted` para obter uma tabela ordenada por ordem decrescente de pontos.

   c) Acrescente os argumentos adequados à função `sorted` para obter uma tabela ordenada por ordem decrescente da diferença de golos marcados e sofridos.

   d) Acrescente os argumentos adequados à função `sorted` para ordenar a tabela por ordem decrescente de pontos e, se iguais, pela diferença de golos.

3. Faça uma função que calcule a mediana de uma lista de valores, sem modificar a lista. Se a lista tiver um número ímpar de valores, a mediana é o valor no meio da lista ordenada. Se a lista tiver um número par de valores, a mediana é a média dos dois valores no meio da lista ordenada. (CodeCheck median)

4. O programa `multsort.py` lê uma lista de notas de vários alunos do ficheiro `grades.json` para `lst1` e mostra-a em forma de tabela. Cada elemento da lista é também uma lista com [Número, Nota, Nome, Grupo]. Complete o programa para ordenar a lista por ordem decrescente da nota (em `lst2`). Depois ordene o resultado por ordem crescente do grupo (em `lst3`).

   Repare que o resultado final fica ordenado por grupo, mas dentro de cada grupo, mantém a ordem anterior (por nota decrescente). Isso é garantido porque a linguagem Python usa um algoritmo de ordenação estável. Desta forma, podemos usar ordenações sucessivas para obter ordenações por critérios compostos.

5. *O ficheiro `wordlist.txt` contém uma lista de palavras de língua inglesa, por ordem. Leia essas palavras para uma lista e, usando uma função de pesquisa binária (do módulo `bisect`), descubra quantas palavras começam por "ea", sem ter de percorrer tudo. *Sugestão: procure a primeira palavra com "ea" e a primeira com "eb" e subtraia os índices*. E quantas palavras começam por "truo"? Nenhuma? Então qual é a primeira letra, maior do que 'o', que ocorre após um "tru", nas palavras inglesas?

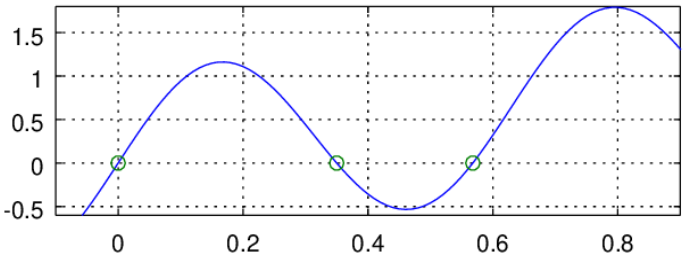
6. *Usando o mesmo princípio, faça uma função que indique todas as letras que podem suceder a um certo prefixo. Pode usar esta função num sistema de escrita inteligente que vai apresentando as letras possíveis para completar um certo prefixo já introduzido. Quando o utilizador introduz mais uma letra, é acrescentada ao prefixo anterior e apresenta-se nova lista de possibilidades e assim sucessivamente.

7. *O programa `insertionSort.py` tem uma implementação do algoritmo de ordenação por inserção. Modifique a função para aceitar um argumento opcional `key=` que funcione tal como na função pré-definida `sorted`.

8. Integração numérica de uma função pela regra dos trapézios. (Codecheck integrate1)

9. *No programa `polynomial.py` pretendemos definir uma função que crie um polinómio de segundo grau arbitrário. Ou seja, `p = polynomial2(2, -1, 3)` deve colocar em `p` uma função tal que `p(x)` calcule o valor $p(x) = 2x^2 - x + 3$ para qualquer valor `x` que lhe seja passado. Note que dentro do corpo de `polynomial2` terá de definir uma nova função para devolver. Essa definição poderá ser feita com uma instrução `def` ou com uma expressão `lambda`.

10. **A função $f(x) = x + sin(10x)$ é uma função contínua (ver gráfico abaixo) e muda de sinal no intervalo [0.2, 0.4], já que $f(0.2) > 0 \wedge f(0.4) < 0$. Assim, pelo teorema de Bolzano, sabemos que tem pelo menos uma raiz (um zero) nesse intervalo.



Implemente uma função `findZero(func, a, b, tol)` que ache um zero de uma função `func` usando o *método da bissecção*. Trata-se de um método de aproximações sucessivas que, em cada iteração, vai reduzindo o intervalo `[a, b]` que contém a raiz para metade. Quando a amplitude do intervalo for inferior à tolerância pretendida, `tol`, paramos e devolvemos `[a, b]`. Isto é uma forma de pesquisa binária, mas em vez de pesquisar numa lista, tem de "pesquisar" numa função real de variável real.