TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 13 (30 minutos) Turno 5ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 12 - Estruturas lineares

Usando pilhas como entidades imutáveis, com as seguintes operações:

```
def nova_pilha():
    return []

def empurra(pilha, elemento):
    return [elemento] + pilha

def topo(pilha):
    if not pilha:
        raise ValueError ('topo: a pilha não tem elementos')
    else:
        return pilha[0]

def tira(pilha):
    if not pilha:
        raise ValueError ('tira: a pilha não tem elementos')
    else:
        return pilha[1:]

def e_pilha(x):
    return isinstance(x, list)

def pilha_vazia(pilha):
    return pilha == []

def pilhas_iguais(p1, p2):
    return p1 == p2

def mostra_pilha(pilha):
    for e in pilha:
        print(' '+str(e))
    print('===')
```

Escreva uma função que recebe uma cadeia de carateres e determina se esta corresponde a um palíndromo (uma palavra ou grupo de palavras em que o sentido é o mesmo, quer se leia da esquerda para a direita quer da direita para a esquerda) . A sua função deve respeitar as barreiras de abstração. Por exemplo,

```
>>> palindromo('amor a roma')
True
>>> palindromo('sopas')
False
>>> palindromo('abcdedcba')
True
```

Solução:

```
def palindromo(cadeia):
    cmp = len(cadeia)
    pilha = nova_pilha()
    for i in range(cmp // 2):
        pilha = empurra(pilha, cadeia[i])
    # decisão de saltar o não sobre o caráter do meio
    if cmp % 2 == 0:
        inicio = cmp // 2
    else:
        inicio = cmp // 2 + 1
    for i in range(inicio, cmp):
        if topo(pilha) != cadeia[i]:
            return False
        pilha = tira(pilha)
    return pilha_vazia(pilha)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 10 (30 minutos) Turno 4ª feira 08:00-09:30
Nome:
Número:
Data:
Curso:

Capítulo 12 - Estruturas lineares

Usando pilhas como objetos, com as seguintes operações:

```
class pilha:
    def __init__ (self):
        self.p = []

    def empurra(self, elemento):
        self.p = [elemento] + self.p
        return self

    def tira(self):
        if not self.p:
            raise ValueError ('tira: a pilha não tem elementos')
        del(self.p[0])
        return self

    def topo(self):
        if not self.p:
            raise ValueError ('tira: a pilha não tem elementos')
        return self.p[0]

    def pilha_vazia (self):
        return self.p == []

    def __repr__ (self):
        if not self.p:
            return '==='
        rep = ''
        for e in self.p:
            rep = rep + ' ' + str(e) + '\n'
        rep += '==='
        return rep
```

Escreva uma função que recebe uma cadeia de carateres e determina se esta corresponde a um palíndromo (uma palavra ou grupo de palavras em que o sentido é o mesmo, quer se leia da esquerda para a direita quer da direita para a esquerda). A sua função deve respeitar as barreiras de abstração. Por exemplo,

```
>>> palindromo('amor a roma')
True
>>> palindromo('sopas')
False
>>> palindromo('abcdedcba')
True
```

Solução:

```
def palindromo(cadeia):
    cmp = len(cadeia)
    p = pilha()
    for i in range(cmp // 2):
        p.empurra(cadeia[i])
    # decisão de saltar o não sobre o caráter do meio
    if cmp % 2 == 0:
        inicio = cmp // 2
    else:
        inicio = cmp // 2 + 1
    for i in range(inicio, cmp):
        if p.topo() != cadeia[i]:
            return False
        p.tira()
    return p.pilha_vazia()
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 13 (30 minutos) Turno 2ª feira 09:00-10:30
Nome:
Número:
Data:
Curso:

Capítulo 12 - Estruturas lineares

Considere a classe fila com os seguintes métodos:

```
class fila:
      def __init__(self, *fila_inicial):
            self.f = list(fila_inicial)
      def inicio (self):
      if not self.f:
                  raise ValueError ('inicio: fila vazia')
            else:
                  return self.f[0]
      def comprimento(self):
            return len(self.f)
      def coloca(self, elemento):
            self.f = self.f + [elemento]
            return self
      def retira (self):
            if not self.f:
                  raise ValueError ('retira: fila vazia')
            else:
                  del(self.f[0])
      def fila_vazia(self):
      return self.f == []
      def filas_iguais(self, outra):
            return self.f == outra.f
      def __repr__(self):
            f = '< '
            for e in self.f:
                  f = f + e.__repr__() + ' '
            f = f + '<'
            return f
```

Escreva a função *soma_quadrados* que recebe um argumento do tipo fila, cujos elementos são inteiros, e que calcula a soma dos quadrados dos elementos da fila. A sua função não deve destruir a fila recebida. Por exemplo,

```
>>> f = fila(3, 4, 5, 6)
>>> f
< 3 4 5 6 <
>>> soma_quadrados(f)
86
>>> f
< 3 4 5 6 <
```

Resposta:

```
def soma_quadrados(f):
      soma = 0

      for i in range(f.comprimento()):
            e = f.inicio()

            f.retira()
            f.coloca(e)
            soma = soma + e ** 2


      return soma
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 13 (30 minutos) Turno 2ª feira 10:30-12:00
Nome:
Número:
Data:
Curso:

Capítulo 12 - Estruturas lineares

Considere a classe fila com os seguintes métodos:

```
class fila:
      def __init__(self, *fila_inicial):
            self.f = list(fila_inicial)
      def inicio (self):
      if not self.f:
                  raise ValueError ('inicio: fila vazia')
            else:
                  return self.f[0]
      def comprimento(self):
            return len(self.f)
      def coloca(self, elemento):
            self.f = self.f + [elemento]
            return self
      def retira (self):
            if not self.f:
                  raise ValueError ('retira: fila vazia')
            else:
                  del(self.f[0])
      def fila_vazia(self):
      return self.f == []
      def filas_iguais(self, outra):
            return self.f == outra.f
      def __repr__(self):
            f = '< '
            for e in self.f:
                  f = f + e.__repr__() + ' '
            f = f + '<'
            return f
```

Escreva a função lista_pares que recebe um argumento do tipo fila, cujos elementos são inteiros, e que devolve a lista com os elementos pares da fila, do início para o fim da fila. A sua função não deve destruir a fila recebida. Por exemplo,

```
>>> f = fila(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
>>> f
< 1 2 3 4 5 6 7 8 9 10 <
>>> lista_pares(f)
[2, 4, 6, 8, 10]
>>> f
< 1 2 3 4 5 6 7 8 9 10 <
```

Resposta:

```
def lista_pares(f):
    lst = []

    for i in range(f.comprimento()):
        e = f.inicio()
        f.retira()
        f.coloca(e)
        if e % 2 == 0:
           lst = lst + [e]
    return lst
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 13 (30 minutos) Turno 6ª feira 12:30-14:00
Nome:
Número:
Data:
Curso:

Capítulo 12 - Estruturas lineares

Considere a classe fila com os seguintes métodos:

```
class fila:
      def __init__(self, *fila_inicial):
            self.f = list(fila_inicial)
      def inicio (self):
      if not self.f:
                  raise ValueError ('inicio: fila vazia')
            else:
                  return self.f[0]
      def comprimento(self):
            return len(self.f)
      def coloca(self, elemento):
            self.f = self.f + [elemento]
            return self
      def retira (self):
            if not self.f:
                  raise ValueError ('retira: fila vazia')
            else:
                  del(self.f[0])
      def fila_vazia(self):
      return self.f == []
      def filas_iguais(self, outra):
            return self.f == outra.f
      def __repr__(self):
            f = '< '
            for e in self.f:
                  f = f + e.__repr__() + ' '
            f = f + '<'
            return f
```

Escreva a função `estatisticas` que recebe um argumento do tipo fila, cujos elementos são inteiros, e que devolve uma cadeia de carateres com o número de elementos na fila, o elemento mínimo, a média e o elemento máximo. A sua função não deve destruir a fila recebida. Por exemplo,

```
>>> f = fila(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
>>> f
< 1 2 3 4 5 6 7 8 9 10 <
>>> estatisticas(f)
'N: 10, Min: 1, Avg: 5.5, Max: 10'
>>> f
< 1 2 3 4 5 6 7 8 9 10 <
```

Solução:

```
def estatisticas(f):
    n = f.comprimento()
    min,max,soma = f.inicio(), f.inicio(), 0
    for i in range(n):
        e = f.inicio()
        f.retira()
        f.coloca(e)
        if e > max: max = e
        if e < min: min = e
        soma = soma + e
    return 'N: {}, Min: {}, Avg: {}, Max: {}'.format(n,min,soma/n,max)
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 13 (30 minutos) Turno 6ª feira 14:00-15:30
Nome:
Número:
Data:
Curso:

Capítulo 12 - Estruturas lineares

Considere a classe fila com os seguintes métodos:

```
class fila:
      def __init__(self, *fila_inicial):
            self.f = list(fila_inicial)
      def inicio (self):
      if not self.f:
                  raise ValueError ('inicio: fila vazia')
            else:
                  return self.f[0]
      def comprimento(self):
            return len(self.f)
      def coloca(self, elemento):
            self.f = self.f + [elemento]
            return self
      def retira (self):
            if not self.f:
                  raise ValueError ('retira: fila vazia')
            else:
                  del(self.f[0])
      def fila_vazia(self):
      return self.f == []
      def filas_iguais(self, outra):
            return self.f == outra.f
      def __repr__(self):
            f = '< '
            for e in self.f:
                  f = f + e.__repr__() + ' '
            f = f + '<'
            return f
```

Escreva a função `calcula_serie` que recebe um argumento do tipo fila, cujos elementos são inteiros, e que devolve uma cadeia de carateres com o número de elementos na fila, o elemento mínimo, a média e o elemento máximo. A sua função não deve destruir a fila recebida. Por exemplo,

```
>>> f = fila(1, 2, 3, 4, 5)
>>> f
< 1 2 3 4 5 6 7 8 9 10 <
>>> calcula_serie(lambda x: x**3, f)
225
>>> f
< 1 2 3 4 5 6 7 8 9 10 <
```

Solução:

```
def calcula_serie(func, f):
   soma = 0
   for i in range(f.comprimento()):
      e = f.inicio()
      f.retira()
      f.coloca(e)
      soma += func(e)
   return soma
```

TÉCNICO LISBOA

Fundamentos de Programação - 2019/2020 Aula Prática 13 (30 minutos) Turno 6ª feira 15:30-17:00
Nome:
Número:
Data:
Curso:

Capítulo 12 - Estruturas lineares

Usando pilhas como objetos, com as seguintes operações:

```
class pilha:
    def __init__ (self):
        self.p = []

    def empurra(self, elemento):
        self.p = [elemento] + self.p
        return self

    def tira(self):
        if not self.p:
            raise ValueError ('tira: a pilha não tem elementos')
        del(self.p[0])
        return self

    def topo(self):
        if not self.p:
            raise ValueError ('tira: a pilha não tem elementos')
        return self.p[0]

    def pilha_vazia (self):
        return self.p == []

    def __repr__ (self):
        if not self.p:
            return '==='
        rep = ''
        for e in self.p:
            rep = rep + ' ' + str(e) + '\n'
        rep += '==='
        return rep
```

Escreva uma função que recebe uma cadeia de caracteres e determina se esta corresponde a um número *capicua* (um número que se lê igualmente da direita para a esquerda ou vice-versa). A sua função deve respeitar as barreiras de abstração. Por exemplo,

```
>>> capicua('123454321')
True
>>> capicua('12345')
False
>>> capicua('657756')
True
```

Solução:

```
def capicua(cadeia):
    cmp = len(cadeia)
    p = pilha()
    for i in range(cmp // 2):
        p.empurra(cadeia[i])
    # decisão de saltar o não sobre o caráter do meio
    if cmp % 2 == 0:
        inicio = cmp // 2
    else:
        inicio = cmp // 2 + 1
    for i in range(inicio, cmp):
        if p.topo() != cadeia[i]:
            return False
        p.tira()
    return p.pilha_vazia()
```