Big Data & Data Science

Estruturas de dados em Python

Revisão de funções

Entrada, instruções/corpo, **VALOR DE RETORNO**

- Saída **DEVE** retornar um resultado para quem chamou a função, caso esperado:
  - Ex.: def imprimeRefrao()
    - Imprime com *print* sem retornar
    - Chamada → imprimeRefrao
  - Ex.: def fatorial(n)
    - Pode imprimir, mas pode ter que devolver se chamada:
      - fat = fatorial(n)
  - Variável “fat” espera um valor de retorno!
  - Saída == comando **return**

Revisão de funções

ENTRADA, corpo/instruções, valor de retorno (saída)

- def func(ENTRADA), ENTRADA pode ser 0 ou mais “variáveis”
  - Argumentos podem ser mandatórios ou opcionais

```
def pergunta(confirma, tentativas = 3, lembrete = “Tente novamente!”):
    while True:
        ok = input(confirma)
        if ok in (‘s’, ‘S’):
            return True
        if ok in (‘n’, ‘N’):
            return False
        tentativas = tentativas – 1
        if tentativas < 0:
            raise ValueError(“Resposta inválida!”)
        print(lembrete)
```

Revisão de funções

Argumento mandatório:

```
def pergunta(confirma, tentativas = 3, lembrete = “Tente novamente!”):
    while True:
        ok = input(confirma)
        if ok in (‘s’, ‘S’):
            return True
        if ok in (‘n’, ‘N’):
            return False
        tentativas = tentativas – 1
        if tentativas < 0:
            raise ValueError(“Resposta inválida!”)
        print(lembrete)
```

- Chamada: pergunta(“Deseja mesmo apagar?”)

Revisão de funções

Argumento mandatório e valores *default*:

```
def pergunta(confirma, tentativas = 3, lembrete = “Tente novamente!”):
    while True:
        ok = input(confirma)
        if ok in (‘s’, ‘S’):
            return True
        if ok in (‘n’, ‘N’):
            return False
        tentativas = tentativas – 1
        if tentativas < 0:
            raise ValueError(“Resposta inválida!”)
        print(lembrete)
```

- Chamada: pergunta(“Deseja mesmo apagar?”)

Revisão de funções

1 argumento opcional:

```
def pergunta(confirma, tentativas = 3, lembrete = “Tente novamente!”):
    while True:
        ok = input(confirma)
        if ok in (‘s’, ‘S’):
            return True
        if ok in (‘n’, ‘N’):
            return False
        tentativas = tentativas – 1
        if tentativas < 0:
            raise ValueError(“Resposta inválida!”)
        print(lembrete)
```

- Chamada: pergunta(“Deseja mesmo apagar?”, 2)

Revisão de funções

Todos os argumentos:

```
def pergunta(confirma, tentativas = 3, lembrete = “Tente novamente!”):
    while True:
        ok = input(confirma)
        if ok in (‘s’, ‘S’):
            return True
        if ok in (‘n’, ‘N’):
            return False
        tentativas = tentativas – 1
        if tentativas < 0:
            raise ValueError(“Resposta inválida!”)
        print(lembrete)
```

- Chamada: pergunta(“Deseja mesmo apagar?”, 2, “Por favor, não!”)

Exercício

- Refazer a função fatorial com entrada do usuário!

Exercício

- Refazer a função fatorial com entrada do usuário!
- Resposta:

```
>>> def fat(n):
...     if n == 0: return 1
...     return n*fat(n-1)
...
>>> n = input("Qual n? ")
Qual n? 5
>>> fat(int(n))
```

Composição de funções

Série de Taylor:

- Recriar a função retornando o seno convertido em graus e chamando-a no corpo de um *script*
- Quando e como parar?
- Cada termo da série é:

```
y += ((-1)**k)*(x**(1+2*k))/math.factorial(1+2*k)
```

Iterações

Laço + critério de parada

- Como seria uma função para calcular uma série de Taylor com *n* iterações?

- Como seria uma função para calcular uma série de Taylor com precisão *x*?

Iterações

Como seria uma função para calcular uma série de Taylor com *n* iterações?

```
import math

def deg2rad(graus):
  return graus/180*math.pi

def taylor(x, n):
  y = 0
  for k in range(n):
    y += ((-1)**k)*(x**(1+2*k))/math.factorial(1+2*k)
  return y


x = float(input("Angulo em graus: "))
rad =
print("Seno de %d = %f" %(x, taylor(deg2rad(x),int(input("Iteracoes? "))))  )
```

Iterações

E calcular uma série de Taylor com precisão *x*?

```
(...)

def taylor(x, prec):
  y, ya = 0.0, 0.0
  k = 0
  while True and k < 10:
    y += ((-1)**k)*(x**(1+2*k))/math.factorial(1+2*k)
    k += 1
    if (math.fabs(y-ya) <= prec): return y
    ya = y
  return y

x = float(input("Angulo em graus: "))
print("Valor do Seno de %d aproximadamente = %f" %(x, taylor(deg2rad(x),float(input("Precisão? "))))  )
```

Fibonacci

Fazer função que:

- Dado “n”, retorna o enésimo número de Fibonacci

Fibonacci

Fazer função que:

- Dado “n”, retorna o enésimo número de Fibonacci

```
def fibo(n):
  a,b, fibo = 0,1,0
  if (n <= 1):
    return 0
  if (n == 2):
    return 1
  for i in range(2,n):
    fibo = a + b
    a = b
    b = fibo
  return fibo

n = int(input("n-esimo num. de Fibonacci: "))
print("%d fibo = %d" %(n, fibo(n)))
```

Fibonacci

Fazer função que:

- Dado “n”, retorna o último número de Fibonacci <= n

Fibonacci

Fazer função que:

- Dado “n”, retorna o último número de Fibonacci <= n

```
def fibo(n):
  a,b = 0,1
  if (n <= 1):
    return 0
  if (n == 2):
    return 1
  while b < n:
    a, b = b, a+b
  return a

n = int(input("Num. de Fibonacci até "))
print("fibo < %d = %d" %(n, fibo(n)))
```

Fibonacci

Fazer função que:

- Imprime uma sequência até “n”

```
>>> a, b = 0, 1
>>> while b < 1000:
...       print(b, end=' ')
...       a, b = b, a+b
...
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Fibonacci

Fazer função que:

- Retorna uma sequência de Fibonacci
- =O

Fibonacci

Fazer função que:

- Retorna uma sequência de Fibonacci

```
def fibo(n):
  a, b = 0, 1
  fibo = [0, 1]
  if n > 2:
    for i in range(2,n):
      fibo.append(a+b)
      a, b = b, a+b
  return fibo

print(fibo(int(input())))
```

```
$ python fibo3.py
7
[0, 1, 1, 2, 3, 5, 8]
```

Estruturas de dados

Servem para abstrair elementos do mundo real

- “Convertem” elementos para que um computador os entenda
- Como um médico enxerga um paciente?
  - Diversas abstrações possíveis, ex.: cirurgião
  - Um amontoado de células? Órgãos? Partes de um órgão?
- São criadas de acordo com a necessidade do problema a ser resolvido → níveis de abstração

Estruturas de dados

Níveis de abstração: problema da classificação da íris

- Uma flor é representada por alguns atributos:
  - Comprimento da sépala em cm
  - Largura da sépala em cm
  - Comprimento da pétala em cm
  - Largura da pétala em cm
  - Classe: {setosa, versicolour, virginica}

Estruturas de dados

Níveis de abstração para diferentes problemas:

- Como representar um e-mail para verificar se é SPAM ou não?
- Como representar uma cidade?
- Como representar o clima de uma região?
- Como representar um ser humano?

Estruturas de dados

Como representar um elemento qualquer?

- Tuplas
  - Data = (dia, mês, ano, [descrição])
  - Humano = (“nome”, idade, peso, altura, “sexo”)

- Estruturas de dados **criam** novos tipos de dados, compostos!

- Crie variáveis do tipo Data acima para as aulas restantes do curso de Python

Estruturas de dados

Crie variáveis do tipo Data acima para as aulas restantes do curso de Python:

- Como armazenar todas elas em uma váriavel de tuplas representando um **calendário**?

Listas em Python

Numero = [3, 2, 8, 25, 0]

- A variável “Numero” é uma **lista**!
- Lista pode ser qualquer coisa:

  UmaLista = [1, "batata", (1.73, 61.0)]

- Tamanho da lista: len(UmaLista) → 3

Listas em Python

Métodos do tipo lista (notação “ponto”):

- Criar lista vazia

```
numero = []
```

-
```
>>> numero
[]
```

Listas em Python

Métodos do tipo lista (notação “ponto”):

- Criar lista com elementos

```
numero = [3, 2, 8, 25, 0]
```

-
```
>>> numero
[3, 2, 8, 25, 0]
```

Listas em Python

Métodos do tipo lista (notação “ponto”):

- Inserir no final da lista → append(<ELEMENTO>)

```
numero.append(77)
```

-
```
>>> numero
[3, 2, 8, 25, 0, 77]
```

Listas em Python

Métodos do tipo lista (notação “ponto”):

- Inserir em posição específica → insert(pos, elem)

```
numero.insert(1, -13)
```

-
```
>>> numero
[3, -13, 2, 8, 25, 0, 77]
```

Listas em Python

Métodos do tipo lista (notação “ponto”):

- Remover elemento (um só) → remove(x)

```
numero.remove(25)
```

-
```
>>> numero
[3, -13, 2, 8, 0, 77]
```

Listas em Python

Métodos do tipo lista (notação “ponto”):

- Remover elemento em posição específica → pop([pos])

```
numero = [3, -13, 2, 8, 0, 77]
>>> numero.pop(3)
8
>>> numero.pop()
77
```

Lista como um vetor

- Numero[0] = primeira posição
- Numero[1] = segunda posição
- ...
- Numero[-1] = última posição

- Numero[-2] = ?
- Numero[-2:] = ?
- Numero[:] = ?

Lista como uma matriz

```
>>> coisas = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> coisas
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> coisas[1]
[4, 5, 6]
>>> coisas[2][2]
9
```

Lista de listas (tipos distintos)

```
>>> frutas = [“banana”, “maçã”]
>>> vestes = ["bota", "sapato", "chinelo"]
>>> coisas.append(frutas)
>>> coisas.append(vestes)

>>> coisas
[[1, 2, 3], [4, 5, 6], [7, 8, 9], ['banana', 'maçã'], ['bota', 'sapato', 'chinelo']]
```

Lista de listas

UNIÃO:

```
>>> coisas
[[1, 2, 3], [4, 5, 6], [7, 8, 9], ['banana', 'maçã'], ['bota', 'sapato', 'chinelo']]
```

```
>>> coisas[1] + coisas[3]
[4, 5, 6, 'banana', 'maçã']
```

Laços revisitados

Alguns tipos possíveis de *for*:

- Range:
  - For i in range(x) ← Já vimos!
  - For i in range(len(LISTA))
- Objeto:
  - For i in LISTA

Por falar na função len()

Escreva uma função em que, dada uma palavra, justifique esta palavra à direita do parágrafo na coluna 80.

- Ex.: justificaD(palavra)

<palavra>

Por falar no operador “in”

- for letra in “palavra”
  - frase = “alguma coisa”
  - frase[0] = ‘a’
- for elemento in Lista

Matriz de zeros

Dada uma lista Z = [0, 0, 0], como seria uma matriz 3x3 de zeros?

Matriz de zeros

Dada uma lista Z = [0, 0, 0], como seria uma matriz 3x3 de zeros?

- M0.append(Z)
- M0.append(Z)
- M0.append(Z)

- Pode isso?

Alocação dinâmica de memória

Cuidado com os ponteiros ocultos de Python!!!

- >>> a = [0, 0, 0]
- >>> b = a
- >>> b
- [0, 0, 0]
- >>> b[1] = 1
- >>> b
- [0, 1, 0]
- >>> a
- [0, 1, 0]

Alocação dinâmica de memória

Como funcionam as variáveis (referências)

```
>>> a = [1, 2, 3]
>>> b = a

>>> a is b
True
```

Alocação dinâmica de memória

Como funcionam as variáveis (referências)

```
>>> a.append(4)

>>> b
[1, 2, 3, 4]
```

Alocação dinâmica de memória

Como funcionam as variáveis (referências)

```
>>> id(a)
140064975855496

>>> id(b)
140064975855496
```

Cópia superficial (*slicing*)

```
>>> c = []
>>> c.append(a[:])
>>> c.append(b[:])
>>> c
[[2, 1, 0], [2, 1, 0]]

>>> c[0][0] = 0
>>> c
[[0, 1, 0], [2, 1, 0]]
```

Cópia superficial (*copy*)

```
>>> import copy
>>> b = copy.copy(a)
>>> a.append(5)
>>> a
[1, 2, 3, 4, 5]
>>> b
[1, 2, 3, 4]
```

Cópia superficial em lista de listas

```
>>> a = [[0, 0], [0, 0]]
>>> b = copy.copy(a)
>>> a[0][0] = 1
>>> b
[[1, 0], [0, 0]]
```

???

Cópia profunda

```
>>> a = [[0, 0], [0, 0]]
>>> b = copy.deepcopy(a)
>>> a[0][0] = 1
>>> b
[[0, 0], [0, 0]]
```

YAY!!!

Exercícios

Dada a lista passada pelo professor:

- Criar uma função que ordene a lista alfabeticamente
- Criar uma função que inverta a ordem dos elementos da lista
- Vimos que UNIÃO de listas se faz com o operador +
  - Faça uma função em que, dadas 2 listas, retorne a intersecção
- Criar uma função que conte os elementos repetidos da lista

Intersecção de listas

Lists comprehension:

- >>> a = [1, 2, 3]
- >>> b = [1, 4, 3]
  - [x for x in a if x not in b]
- [1, 3]

O Dicionário!

- Criar uma função que conte os elementos repetidos da lista
- Dic = {}
  - Composto por “chave” e “valor”
    - 'guarda-roupa': 4
    - 'televisão': 1
    - 'guarda-treco': 2
    - 'cinto': 4 …

```
>>> for chave in dic:
...     print(chave)
...
guarda-roupa
televisão
guarda-treco
cinto
pincel
```

```
>>> for chave in dic:
...
print(dic[chave])
...
4
1
2
4
16
```

Pilhas

O que é?

- Estrutura de dados com um atributo “topo”
- Política LIFO
  - Último a chegar, primeiro a sair
- Métodos:
  - PUSH para inserir no topo
  - POP para remover do topo

Filas

O que é?

- Estrutura de dados com atributos “início” e “fim”
- Política FIFO
  - Primeiro que chega, primeiro que sai
- Métodos:
  - ENFILEIRA no fim
  - DESENFILEIRA do início

Pilha em Python

Usando uma lista, temos:

- Pilha = [1, 2, 3]
- Pilha.append(4) → faz a função do PUSH, pois insere no fim
- Pilha.pop() → remove elemento do fim, logo segue a LIFO

Fila em Python

Usando uma lista, temos:

- Fila = [1, 2, 3]
- Fila.append(4) → insere no fim da fila
- Fila.pop(0) → remove do início, logo obedece a FIFO

Vetor em Python

Diferença entre vetor e lista?

- import array
  - meuVetor = array.array('i', [0, 1, 2])

Type code	C Type	Python Type	Minimum size in bytes	Notes
'b'	signed char	int	1	
'B'	unsigned char	int	1	
'u'	Py_UNICODE	Unicode character	2	(1)
'h'	signed short	int	2	
'H'	unsigned short	int	2	
'i'	signed int	int	2	
'I'	unsigned int	int	2	
'l'	signed long	int	4	
'L'	unsigned long	int	4	
'q'	signed long long	int	8	(2)
'Q'	unsigned long long	int	8	(2)
'f'	float	float	4	
'd'	double	float	8	

Vetor em Python

Outros métodos em:

- https://docs.python.org/3.6/library/array.html

Estruturas de dados “custom”

Para criar uma estrutura de dados como uma *struct* em C, deve-se usar classes. Será visto oportunamente...

Exercícios

- Inverter uma lista usando conceitos de pilha
- Deduplicar elementos de uma lista
- Função para dado “n”, criar lista de números até “n” contendo os quadrados dos números usando *list comprehension*
- Dadas duas listas, l1 = [1, 2, 3] e l2 = [3, 1, 4], combine os elementos das duas, par a par (em tuplas), sem que uma tupla contenha elementos repetidos usando *list comprehension*

Exercício 1

Inverte lista usando pilha:

```
>>> for i in range(len(l)):
...   j -= 1
...   pilha.append(l[j])
```

Exercício 2

Deduplicar elementos de uma lista

```
>>> l1 =[]
>>> for i in l:
...   if i not in l1:
...     l1.append(i)
...
>>> l1
['guarda-chuva', 'banana', 'guarda-roupa', 'batata', 'sapato', 'pincel', 'Arturito', 'tartaruga',
'guarda-treco', 'colher', 'ovo', 'olho', 'livro', 'televisão', 'guarda', 'uva', 'cabrito', 'piolho', 'cinto']
```

Exercícios 3 e 4

LISTS COMPREHENSION:

- squares = [x**2 for x in range(10)]
- [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]

Abrindo um arquivo

Leitura:

- Modo padrão leitura em modo texto: “r” (*read*)
- open(“arquivo.txt”)

Escrita:

- open(“resultado.txt”, ‘w’)
- ‘a’ → modo *append*, insere no fim do arquivo se este existe

Leitura de arquivos

Função “read()”:

- Lê o arquivo inteiro

```
>>> f.read()
'este é\num arquivo\nde texto\n'
```

Função “readline()”:

- Lê uma linha por chamada

```
>>> f.readline()
'este é\n'
>>> f.readline()
'um arquivo\n'
>>> f.readline()
'de texto\n'
```

Lendo uma lista de um arquivo

Lista.txt:

- ['guarda-chuva', 'banana', ..., 'cinto']
- Como eu leio a lista do arquivo diretamente para uma lista?
  - Operações sobre *strings*:
    - lstrip(‘o_que_quero_tirar’) → remove caracter(es) à esquerda
    - rstrip() → remove caracter(es) à direita
    - strip() → remove caracter(es) à esquerda e à direita

Lendo uma lista de um arquivo

```
>>> a = f.readline().rstrip("\n")
>>> a
'um arquivo'
```

```
>>> palavra = ".dados."
>>> palavra.strip('.')
'dados'
```

```
>>> "www.dominio.com".lstrip("w.")
'dominio.com'
```

Lendo uma lista de um arquivo

```
lista = eval(open("lista.txt").read())
```

- EVAL?
  - Avalia uma *string* como uma expressão Python
  - Lida com comandos e estruturas, como as listas
  - =)

Lendo um arquivo em uma lista

Função “readlines()”

- Lê todas as linhas, linha a linha, em uma LISTA!

```
>>> f.readlines()
['este é\n', 'um arquivo\n', 'de texto\n']
```

- Se eu quiser ler sem a quebra de linha:

```
>>> f = open("ex.txt").read().splitlines()
>>> f
['este é', 'um arquivo', 'de texto']
```

ou

```
[line.rstrip('\n') for line in f]
```

Lendo um arquivo em uma lista

Para ler um arquivo de texto e separar por palavras:

- arq = open("texto.txt")>>> arq = open("texto.txt")
- >>> arq
  - <_io.TextIOWrapper name='texto.txt' mode='r' encoding='UTF-8'>
- >>> listaPalavras = arq.read().**split()**

A função *split()*

Separa uma *string* por um caracter delimitador:

- cdatetime,district,code,latitude,longitude
- 1/1/06 0:00,3,2404,38.55042047,-121.391415

```
>>> f = open("teste.csv")
>>> for line in f:
...   line.strip("\n").split(",")
...
['cdatetime', 'district', 'code', 'latitude', 'longitude']
['1/1/06 0:00', '3', '2404', '38.55042047', '-121.3914158']
```

O módulo “SYS”

- Import sys
- argv é a lista de argumentos da linha de comando passados como entrada
  - argc em Python é simplesmente o comprimento de argv
- https://docs.python.org/3.5/library/sys.html

Exercícios

Dado o arquivo “SacramentocrimeJanuary2006.csv”, fazer um *script* que o lê e:

- Apresente a quantidade de crimes por data
- Calcule a porcentagem dos crimes por tipo
- Mostre a média de crimes por mês em geral
- Mostre a média de crimes por mês por distrito

Exercícios

No arquivo “Sacramentorealestatetransactions.csv”:

- Quantos e quais são os tipos distintos de imóveis?
- Quantos imóveis estão à venda de cada tipo?
- Quais as médias de banheiros, quartos e metros quadrados?
- Quais CEPs têm os imóveis mais baratos e quais mais caros?
- Qual o preço médio dos imóveis em Sacramento?

Exercícios

No arquivo “SalesJan2009.csv”:

- Quantos países estão listados?
- Quantos tipos de pagamento?
- Crie um dicionário onde cada chave é um país e o valor é uma lista de estruturas contendo o tipo de pagamento e a frequência com a qual esse tipo foi usado. **Ex.: No Brasil, usou-se dinheiro *x* vezes, boleto *y* vezes, ...**

Exercícios

Com base nos atributos listados em algum dos CSVs dados (*Moodle*):

- Crie uma estrutura de dados que o represente
- Use as estruturas aprendidas em sala
  - Listas
  - Dicionários
  - Tuplas
  - O que mais quiser utilizar...