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]
```