ACH5531

Introdução à Computação

Funções e Gerenciamento de Arquivos

Prof. Dr. Grzegorz Kowal

grzegorz.kowal@usp.br

https://sites.google.com/usp.br/ach5531

1° sem 2019 – sexta-feira, 14h00-15h45 – CB, Bloco 3, 2° andar, Lab. 6

Funções

Funções são blocos de instrução que podem ser invocados de qualquer parte do nosso código. Toda função, por definição, possui um nome, pode receber parâmetros e pode retornar valores. Nem toda função receberá parâmetros, da mesma forma, que nem toda função retornará valores. Tudo dependerá da situação, ainda assim, a estrutura sempre seguirá um padrão e por isso, é facilmente reconhecido.

Na linguagem Python existem várias funções predefinidas, como input(), print(), int(), float(), str(), type(), len(), etc.

Como definir uma função

A sintaxe de uma função é definida por três partes: nome, parâmetros e corpo, o qual agrupa uma sequência de linhas que representa algum comportamento. No código abaixo, temos um exemplo de declaração de função em Python:

Exemplo 1:

```
def hello(nome):
  print("Ola ", nome)
```

Essa função, de nome **hello**, tem como objetivo imprimir o nome que lhe é passado por parâmetro (também chamado de argumento). A palavra reservada **def**, na primeira linha, explicita a definição da função naquele ponto. Em seguida, entre parênteses, temos o parâmetro *nome*. Ainda na mesma linha, observe a utilização dos dois pontos (:), que indicam que o código identado nas linhas abaixo faz parte da função que está sendo criada. Aqui, é importante ressaltar que, para respeitar a sintaxe da linguagem, a linha 2 está avançada em relação à linha 1.

Como executar uma função

Para executar a função, de forma semelhante ao que ocorre em outras linguagens, devemos simplesmente chamar seu nome e passar os parâmetros esperados entre parênteses, conforme o código a seguir.

Exemplo 2:

```
>>> hello("Pedro")
Ola Pedro
```

Caso seja necessário, também é possível definir funções com nenhum ou vários argumentos, como no código abaixo:

Exemplo 3:

```
def hello(nome, idade):
  print("Ola ", nome, "\nSua idade e", idade)

>>> hello("Pedro", 29)
Ola Pedro
Sua idade e 29
```

Função retornando um valor

Assim como podem receber valores de entrada, as funções também podem produzir valores de saída, provenientes de determinadas operações. Nos exemplos anteriores, apenas imprimimos um valor com a função **print**, sem retornar explicitamente um resultado. Abaixo temos uma função que faz uma conta e retorna o valor de resultado:

Exemplo 4:

```
def soma(x, y):
  s = x + y
  return s

>>> soma(1,2)
3
```

A função soma() recebe dois parâmetros, x e y, quais soma e atribuída a variável local s, qual valor é retornado com sintaxe **return** s.

Parâmetros com valores padrões

É possível definir parâmetros de funções com valores padrões, ou opcionais. No caso de um parâmetro com valor padrão, a pessoa que chamar esta função não será obrigada a informar nenhum valor. Porém, caso seja passado algum valor, ele sobrescreverá o valor pré definido.

Exemplo 5:

```
def incrementar(n, i = 1):
  m = n + i
  return m

>>> incrementar(5)
6
>>> incrementar(5, 2)
7
>>> incrementar(3, i = 5)
8
```

Exemplo 6

Escreva um programa que solicita o usuário um número inteiro positivo e imprime o fatorial dele: n! = 1*2*...*n, 0! = 1, 1! = 1. Use uma função em Python para calcular o fatorial de número dado.

```
def fatorial(n):
  f = 1
  i = n
  while i > 0:
    f = f * i
    i = i - 1
  return f

print("Programa imprime fatorial de número inteiro.\n")

while True:
  s = input("Digite o número inteiro positivo: ")
  if s != “”:
    n = int(s)
    print("Fatorial de ", n, “ é “, fatorial(n))
  else:
    break
```

Arquivos

Uma das tarefas mais importantes que realizamos no dia-a-dia é a manipulação de dados gravados em arquivos, dos mais variados tipos. Por exemplo, um administrador de sistemas precisa, com frequência, acessar informações de configuração armazenadas em arquivos de texto, e muitas vezes, efetuar alterações nesses arquivos.

Um **arquivo** é uma área no disco onde gravamos e de onde lemos dados. Um arquivo pode ser de texto simples ou um arquivo binário.

Arquivos de texto (plaintext) são uma sequência estruturada de linhas, que contém cada uma uma sequência de caracteres, sem informações sobre formatação.

Cada linha em um arquivo de texto é terminada com um caractere especial de fim de linha, EOL (End of Line). No geral, é o caractere newline (**\n**). Esse caractere finaliza a linha atual e diz ao interpretador que uma nova linha se inicia.

Já um arquivo binário é qualquer arquivo que não seja arquivo de texto padrão. Os arquivos binários somente devem ser processados por uma aplicação que compreenda a estrutura do arquivo. Como exemplos, temos arquivos pdf, doc, imagens, executáveis, mp3 e planilhas.

Arquivos em Python

Para trabalharmos com arquivos no Python usaremos o objeto **file**.

Os objetos file contém métodos e atributos que podem ser usados para coletar informações e manipular um arquivo.

Um objeto file possui um atributo **nome**, que é o nome do arquivo a ser manipulado, e um atributo **modo**, que é o modo como o arquivo será acessado.

Os modos de acesso a um arquivo disponíveis em Python estão listados na tabela a seguir:

Modo	Tipo de acesso
r	Somente leitura
w	Escrita, apagando (truncando) o conteúdo existente no arquivo
a	Escrita, preservando o conteúdo existente (append). O arquivo é criado, se não existir. O texto é inserido no final do arquivo.
b	Modo binário
+	Abre o arquivo para atualização – leitura e escrita
x	Abre o arquivo para criação exclusiva, falhando se o arquivo já existir.
t	Modo de texto (padrão)

Arquivos em Python

Podemos combinar os modos, por exemplo r+, w+, w+b. O modo padrão é r (leitura) de texto, caso não seja especificado.

Para trabalhar com arquivos de texto precisamos efetuar as seguintes operações, em sequência:

- Abrir o arquivo no modo desejado
- Chamar o método apropriado (leitura/escrita)
- Executar o processamento dos dados do arquivo
- Fechar o arquivo.

A função **open**() retorna um objeto *manipulador* de arquivos, que é um objeto iterável usado para realizar operações sobre um arquivo. Sintaxe:

```
manipulador = open(arquivo, modo)
```

arquivo é um texto que indica o nome / caminho do arquivo a ser aberto no sistema de arquivos.

modo é opcional, sendo que o padrão é t (modo de texto).

Arquivo como entrada de dados

Quando um arquivo é lido, o Python mantém um registro da posição atual de leitura no arquivo. Se ele for lido com o método **read**(), a posição será sempre a do final do arquivo, pois esse método retorna o arquivo inteiro de uma vez. Caso a função **read**() seja chamada novamente, logo na sequência, uma string vazia será retornada, pois não há mais dados a serem lidos na posição atual no arquivo.

Exemplo 7:

```
f = open('arquivo.txt')
print(f.read())
f.close()
```

Podemos alterar o ponteiro de arquivo por meio do método **seek**(), que recebe como argumento a posição desejada – um offset (deslocamento) em bytes. Também podemos descobrir a posição atual dentro do arquivo com o método **tell**().

Podemos usar o método **read**() para ler um número determinado de caracteres, bastando para isso passar com argumento a quantidade de bytes a ser lida.

Arquivo como entrada de dados

Além do método **read**() também podemos ler o conteúdo de um arquivo usando os métodos **readline**() e **readlines**().

readline() - retorna uma linha do texto a cada chamada, na ordem em que aparecem no arquivo (um ponteiro de linha é incrementado a cada nova chamada ao método).

readlines() - retorna uma lista de valores de string do arquivo, sendo que cada string corresponde a uma linha do texto.

Exemplo 8:

```
f = open('arquivo.txt')
print(f.readline())
for linha in f.readlines():
  print(linha)
f.close()
```

Arquivo como entrada de dados

Ao usar a iteração com laço **for**, não é necessário invocar os métodos **read**(), **readline**() ou **readlines**(). Veja o exemplo a seguir:

Exemplo 9:

```
f = open('arquivo.txt')
for linha in f:
  print(linha.rstrip())
f.close()
```

Uma boa prática é sempre fechar um arquivo após o término das operações sobre ele. Para fechar um arquivo, usamos o método **close**() no objeto, como mostrado nos exemplos.

Também é possível fechar um arquivo automaticamente. Para isso, usamos a declaração **with**, como no exemplo a seguir:

Exemplo 10:

```
with open('arquivo.txt') as f:
  for linha in f:
    print(linha.rstrip())
```

Arquivo como saída de dados

Para escrever texto em um arquivo, ajustamos o modo de abertura do arquivo de acordo e usamos o método **write**() do objeto file, que recebe como argumento o texto a ser inserido no arquivo. Sintaxe:

```
manipulador.write(texto)
```

Veja o exemplo a seguir, usando o modo w:

Exemplo 11:

```
with open('arquivo.txt', 'w') as f:
  f.write('Texto adicionado à primeira linha.\n')
  f.write('Texto que vai na segunda linha.\n')
with open('arquivo.txt') as f:
  print(f.read())
```

Um ponto importante a ser lembrado é de que o método write() não acrescenta o caractere de newline automaticamente, por isso precisamos fornecê-lo junto com o texto (\n).

Arquivo como saída de dados

No exemplo seguinte, escrevemos no arquivo de texto, mas sem utilizar a declaração with. Neste caso, devemos nos lembrar de fechar o arquivo por meio do método close():

Exemplo 12:

```
# Criando e escrevendo em arquivos de texto (modo 'w').
f = open('arq01.txt','w')
f.write("Criando um arquivo de texto com Python\n")
f.close()

# Lendo o arquivo criado:
arquivo = open('arq01.txt','r')
for linha in arquivo:
    print(linha.rstrip())
arquivo.close()
```

Arquivo como saída de dados

Caso o método **write**() seja invocado sobre um arquivo que foi aberto pelo método 'w', o conteúdo do arquivo será substituído pelo novo conteúdo fornecido ao método, pois o método **write**() primeiramente trunca o arquivo (apaga todo o seu conteúdo), e só então permite a gravação de texto novo. Ou seja, ele sobrescreve o conteúdo do arquivo.

Para preservar o conteúdo existente no arquivo, devemos abri-lo usando o modo 'a' (append), que permite acrescentar texto novo ao final do arquivo, sem apagar o conteúdo já existente. Veja o exemplo a seguir:

Exemplo 13:

```
# Acrescentando texto ao arquivo criado,
# usando o modo de acesso 'a'
f = open('arq01.txt','a')
f.write("Novo texto inserido no arquivo\n")
f.close()
```

Exemplo 14

Grava, no arquivo de saída, uma tabela de quadrados e cubos de números inteiros até número fornecido pelo usuário.

```
print("Programa cria uma tabela de n² e n³ no arquivo.\n")

s = input("Digite o máximo número inteiro: ")
m = int(s)

with open("tabela.txt", "w") as f:
  f.write("n\tn²\tn³\n")
  i = 0
  while i <= m:
    f.write("%d\t%d\t%d\n" % (i, i**2, i**3))
    i += 1
```

Exercícios

12. Usando função fatorial, escreva um programa que grava a tabela de fatorial de 10 números inteiros num arquivo como saída. Depois leia o arquivo criado, determina o último número calculado, e em seguida, acrescenta o arquivo com uma tabela com fatoriais de mais 10 números inteiros.

13. Escreva um programa que leia o exemplo de DNA do arquivo ‘dna.txt’ anexado e acha o número de ocorrências e posição de cada ocorrência de sequencia ‘ACC’. O resultado da analise deve ser gravado no arquivo separado.