Preparação para Maratona de Informática
" PYTHON"

Neste material, veremos como abrir arquivos de texto utilizando alguns recursos Python para manipulação de arquivos, listas e formatação de saída.

Como referência, todos arquivos .TXT utilizados devem estar na mesma pasta onde estiver o script .py.

Trabalhando com arquivos

A função open abre um arquivo em modo leitura 'r' (padrão) e retorna um objeto file. Se não conseguir, levantará uma exceção do tipo OSError. O arquivo também pode ser aberto para escritura 'w'. O modo escritura trunca e substitui o conteúdo existente, se este existir. O modo append 'a', faz com que o modo de gravar adicione o conteúdo novo no final do arquivo, sem truncar.

O comando with funciona como um wrapper que fecha o objeto file automaticamente ao finalizar o bloco.

O objeto file possui os métodos read, readline, readlines e write que permitirão ler o arquivo na sua totalidade, uma determinada quantidade de caracteres, uma linha, ou criar uma lista com cada uma das linhas; além de escrever, novamente, para um arquivo de saída.

- Ler um arquivo de entrada na sua totalidade e escrever o resultado no arquivo de saída (truncando):

```
with open('xanadu.txt') as fin:
        texto = fin.read()
with open('out.txt', 'w') as fout:
        fout.write(texto)
```

- Ler um caractere por vez e escrever o resultado no arquivo de saída (adicionando):

```
with open('xanadu.txt') as fin:
        with open('out.txt', 'a') as fout:
        char = fin.read(1)
        while char != "":
                # Do stuff with char.
                fout.write(char)
                char = fin.read(1)
```

- Ler o arquivo na sua totalidade, armazenar o resultado numa lista onde cada elemento é uma string representando uma linha do arquivo. Iterar a lista e escrever cada linha no arquivo de saída (adicionando):

```
with open('xanadu.txt') as fin:
        with open('out.txt', 'a') as fout:
        lines = fin.readlines()
        for line in lines:
                # Do stuff with line.
                fout.write(line)
```

O método split divide uma string usando um caractere separador especificado como argumento (espaço “ “ por padrão). O método strip vai retirar os caracteres especificados como argumento do início e o final da string. A biblioteca string possui uma constante com todos os caracteres considerados como pontuação. Também é possível iterar linha por linha sem usar o método readlines.

- Ler todas as linhas do arquivo de entrada, iterar, separar em palavras (split) e limpar caracteres indesejados (strip). Escrever uma palavra por linha no arquivo de saída:

```
import string

with open('xanadu.txt', 'r') as fin:
        with open('out.txt', 'a') as fout:
        for line in fin:
                words = line.split()
                for word in words:
                word = word.strip(string.punctuation)
                fout.write(word + "\n")
```

Listas

As listas em Python são coleções arbitrárias de objetos. Elas podem ser acessadas pelo índice e disponibilizam vários métodos para sua manipulação.

- Criar uma lista, três formas diferentes:

```
lista = list()
lista = []
lista = [“João”, “Silva”, 34]
```

- Acessar um elemento: `elemento = lista[2]`

- Atualizar um elemento: `lista[1] = “Souza”`

- Adicionar elemento: `lista.append(objeto)`

- Remover elemento:

```
lista.remove(objeto)        # Remove a primeira ocorrência
lista.pop(index)                    # Sem argumento, remove o
                                    # último elemento
```

- Ordenar elementos:

```
lista.sort()
lista.reverse()
```

- Contar elementos: `lista.count(elemento)`

- Inserir elemento: `lista.insert(index, elemento)`

- Obter o índice de um elemento:

```
index = lista.index(elemento)        # Retorna o index da primeira ocorrência
```

- Comprimento de uma lista: `length = len(lista)`

- Iterar uma lista, duas versões:

```
for i in range(len(lista)):
        print(lista[i])

for elemento in lista:
        print(elemento)
```

- Definir critérios de ordenamento:

```
# Criar a classe Aluno
class Aluno:
        def __init__(self, nome, ano, idade):
        self.nome = nome
        self.ano = ano
        self.idade = idade
        def __str__(self):
        return "{:8} {:2} {:2}".format(self.nome,
                        self.ano, self.idade)

# Criar uma lista de objetos Aluno
alunos = [Aluno("João", 5, 10), Aluno("Maria", 2, 7),
                        Aluno("Lucas", 1, 6)]

# Ordenar a lista segundo um dos atributos do objeto Aluno
alunos.sort(key=lambda aluno: aluno.idade)
```

Formatando Saídas

Função print:

- A função print recebe um ou mais argumentos e exibe eles na saída padrão:

```
a = 5
b = “Animal”
print(“Planta”, 7.8, a, b)
```

Planta 7.8 5 Animal

- O signo ‘+’ concatena duas strings:

```
print(b + “Esquisito”)
```

AnimalEsquisito

Formatando strings:

O operador % formata uma string por meio de marcadores que serão preenchidos pelos argumentos dados em forma de tupla. Os marcadores são:

- %s strings
- %c inteiros ou um simples caractere
- %d ou %i inteiros
- %f ponto flutuante
- %e notação científica (ponto flutuante)
- %o números octais
- %x hexadecimais (minúscula)
- %X hexadecimais (maiúscula)

```
print("%s %s" % ("um", "dois"))
```

```
um dois
```

```
print("%d %d" % (1, 2))
```

```
1 2
```

La sintaxe dos marcadores é a seguinte:

```
%[flag][width][.precision]type
```

Margem e alinhamento:

Um número inteiro na posição ‘width’ controla a quantidade de caracteres que ocupará esse elemento dentro da string. Por padrão, o alinhamento será à direita. A ‘flag’ ‘-’ antes do inteiro fará com que o alinhamento seja à esquerda:

```
print("|%10s|" % ("teste"))          # à direita
```

```
|    teste|
```

```
print("|%-10s|" % ("teste"))         # à esquerda
```

```
|teste     |
```

Precisão

Para os números de ponto flutuante, além da largura, podemos controlar também a precisão, a quantidades de casas decimais após o ponto (com arredondamento):

```
print("|%.2f|" % (5.6794092))

|5.68|

print("|%6.2|" % (5.6794092))

|  5.68|
```

Sendo strings, o valor da precisão determinará quantos caracteres serão efetivamente formatados:

```
print("|%5.3s|" % ("teste"))

|  tes|
```

É possível passar os valores de 'width' e 'precision' como argumentos na tupla:

```
print("|%*s||%*.*f|" % (10, "teste", 6, 2, 3.141628))

|     teste||  3.14|
```

Preenchimento

A flag '0' vai completar o valor do número com zeros à esquerda:

```
print("%010d" % (556))
0000000556
print("%010f" % (5.56))
005.560000
```

Sendo um número de ponto flutuante, a quantidade de casas decimais padrão será de 6. Para limitá-las, precisa adicionar o valor da precisão:

```
print("%010.3f" % (5.56))

000005.560
```

Números com sinais

Para formatar números com sinal usa-se a ‘flag’ “+”:

```
print(“%+d, %+10.2f” % (12, -20.3))
```

```
+12,     -20.30
```