

Profa. Dra. Raquel C. de Melo-Minardi
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais



MÓDULO 2 – PROGRAMAÇÃO

Entrada e saída

ENTRADA E SAÍDA

- ▶ **Entrada e saída** são operações de **comunicação de um programa com o mundo externo**
- ▶ Há dois tipos básicos de entrada e saída que um programa pode realizar:
 - ▶ Por meio do teclado (pode ou não ser via **linha de comando**) e da tela
 - ▶ Por meio da escrita e leitura de **arquivos** gravados no disco rígido do computador
- ▶ Em Python, a entrada e saída via linha de comando e tela também se dão através de arquivos

ENTRADA E SAÍDA

- ▶ Em Python, **arquivos estão associados** a dispositivos tais como
 - ▶ discos rígidos
 - ▶ impressoras
 - ▶ teclados
- ▶ Usamos três **arquivos padrão** para tal:
 - ▶ Toda vez que fazemos um `print`, fazemos um `sys.stdout`
 - ▶ Sempre que lemos um dado através do comando `readline` estamos lendo de um arquivo chamado `sys.stdin`
 - ▶ Mensagens de erro (não veremos nesse curso) são enviadas para o arquivo `sys.stderr`

ENTRADA E SAÍDA

- ▶ Os arquivos
 - ▶ `sys.stdin`
 - ▶ `sys.stdout`
 - ▶ `sys.stderr`
- ▶ normalmente estão associados ao teclado e ao *display* do terminal sendo usado

MÓDULO SYS

- ▶ O módulo `sys` do Python permite realizar algumas interações com o **sistema operacional** da máquina
- ▶ Nessa aula veremos como ele pode ser usado para fazer operações de entrada e saída

ARGUMENTOS DA LINHA DE COMANDO

- ▶ Uma das formas que podemos fazer a entrada de dados para o programa do mundo exterior é coletando argumentos ou parâmetros que podem ser passados ao lado do nome do programa no momento de sua execução (chamada):

```
~raquelcm$ python3 programa.py entrada.txt saida.txt
```

- ▶ Neste exemplo estamos passando dois argumentos para o programa, um arquivo de entrada e um arquivo de saída, ambos arquivos de texto
- ▶ Para coletar esses parâmetros vamos utilizar a lista `argv`, do módulo `sys`, assim:

```
import sys
for param in sys.argv:
    print(param)
```

LEITURA DO TECLADO DENTRO DO PROGRAMA

- ▶ Outra forma em que podemos fazer a entrada de dados para o programa do mundo exterior é coletando argumentos em tempo de execução do programa através de interação com o usuário via função `input`:

```
x = input( 'Digite o valor de x: ' )  
print(x)
```

- ▶ Esse método é menos usado devido a Bioinformática normalmente lidar com dados em larga escala
- ▶ Normalmente, usamos mais leituras de arquivos e fazemos pouca ou nenhuma interação com o usuário

ABERTURA DE ARQUIVOS

- ▶ Em Python, você usará a função `open(nome, modo, buffering)` para abrir um arquivo tanto para leitura quanto para escrita sendo que
 - ▶ `nome`: é o nome do arquivo a abrir
 - ▶ `modo`: é o modo de abertura:
 - ▶ `r`: leitura
 - ▶ `w`: escrita
 - ▶ `b`: binário (há também a opção `t` para texto que é a padrão)
 - ▶ `a`: escrita a partir do final (anexar ou *append*)
 - ▶ `+`: leitura e escrita
 - ▶ `buffering` (opcional): indica se memória (buffers) é usada para acelerar as operações

LEITURA DE ARQUIVOS

- ▶ Uma vez que o arquivo tenha sido aberto, procedemos então à leitura dos dados ou a gravação
- ▶ Veja exemplos de como o tratador de arquivo seria processado após a abertura do arquivo

```
import sys
nomeArqEntrada = sys.argv[1] #Primeiro argumento passado via linha de comando
linhas = open(nomeArqEntrada, "rt")
for l in linhas:
    print(l)
```

- ▶ Certifique-se de passar um nome de arquivo existente ou obterá uma mensagem de erro

SOBRE AS QUEBRAS DE LINHAS...

- ▶ Note que, por padrão, a função `print` de Python sempre imprime com quebra de linhas
- ▶ Há duas variações que você pode usar desse código:
 - ▶ Tirar as quebras de linha de cada linha lida do arquivo

```
for l in linhas:  
    l = l.rstrip()  
    print(l)
```

- ▶ Imprimir sem a quebra de linha

```
for l in linhas:  
    print(l, end='')
```

FECHAMENTO DE ARQUIVOS

- ▶ Após terminar de ler ou gravar em um arquivo, ele sempre deve ser fechado através da função `close`
- ▶ O arquivo pode ser fechado imediatamente ter o seu conteúdo copiado para a variável do tipo lista `linhas`

```
import sys
nomeArqEntrada = sys.argv[1] #Primeiro argumento passado via linha de comando
linhas = open(nomeArqEntrada, 'rt')
for l in linhas:
    print(l)
linhas.close()
```

ESCRITA EM ARQUIVOS

- ▶ A escrita de arquivos é feita pela utilização da função `write`
- ▶ O arquivo deve ter sido aberto para escrita (`w` ou `a`)

```
import sys
nomeArqSaida = sys.argv[1] # Primeiro argumento passado via linha de comando
arq = open(nomeArqSaida, 'w')
arq.write( 'ACCGACCGATGCA' )
arq.close( )
```

- ▶ Se o arquivo ainda não existir, será criado

REMOÇÃO DE ARQUIVOS

- ▶ Para remover um arquivo, precisamos usamos a função `remove` do módulo `os`:

```
import os  
os.remove( 'arquivo.txt' )
```

- ▶ Se o arquivo não existir, você obterá um erro
 - ▶ Para evitar tal erro, melhor testar a existência do arquivo:

```
import os  
if os.path.exists( 'arquivo.txt' ):  
    os.remove( 'arquivo.txt' )  
else:  
    print( 'Arquivo inexistente' )
```

REMOÇÃO DE DIRETÓRIOS

- ▶ Para remover um diretório, use a função `rmdir`:

```
import os  
os.rmdir('diretorio')
```

- ▶ Só é possível remover diretórios vazios

OPÇÕES MAIS AVANÇADAS

- ▶ Há opções mais avançadas para leitura e escrita em arquivos que não veremos nesse curso por limitações de tempo e por serem mais específicas
 - ▶ Acredito que a maioria dos casos de necessidade de tratamento de arquivos é coberta pelos métodos básicos abordados aqui
- ▶ Exemplos de operações mais avançadas
 - ▶ Arquivos binários
 - ▶ Leitura e escrita no mesmo arquivo
 - ▶ Leitura e escrita com *buffer*
 - ▶ Escrita no meio do arquivo
- ▶ Se algum desses métodos for necessário ao seu trabalho, recomendamos estudar cada caso em particular