

# Entrada e saída

---

Marco A L Barbosa  
malbarbo.pro.br

Departamento de Informática  
Universidade Estadual de Maringá



# Conteúdo

Introdução

Entrada e saída padrão

Execução de programas no terminal

Arquivos

Parâmetros na linha de comando

Atividades

# Introdução

- Um programa recebe dados de entrada, faz o processamento e produz dados de saída
- No caso dos programas interativos, este processo se repete até que o programa seja encerrado

- Estamos usando o editor mu para executar e interagir com os programas
- Para executar uma função temos que digitar o nome da função e os argumentos
- Veremos como criar “interfaces” independente do editor mu para interagir com os programas

# Entrada e saída padrão

- Python, assim como a maioria das linguagens de programação, oferece funções de entrada e saída associadas com dispositivos padrão
  - A função `input` lê um texto digitado pelo usuário
  - A função `print` exibe um texto na tela

## Entrada e saída padrão

- Exemplo input

```
>>> nome = input('Qual o seu nome? ')
Qual o seu nome? João
>>> nome
'João'
>>> idade = int(input('Qual a sua idade? '))
Qual a sua idade? 21
>>> idade
>>> 21
```

- A função `input` exibe uma mensagem na tela e lê um valor digitado pelo usuário
- A função `input` devolve o texto (string) digitado pelo usuário

## Entrada e saída padrão

- Exemplo print

```
>>> print('Olá pessoal')
Olá pessoal
>>> troco = 12
>>> r = print('Seu troco é', troco, 'reais')
Seu troco é 12 reais
>>> r
>>>
```

- A função print exibe uma mensagem na tela
- A função print não devolve nenhum valor, o texto é escrito diretamente na tela e não como resultado da função

- A função `print` recebe uma quantidade variável de parâmetros
- Um espaço em branco é automaticamente exibido após cada parâmetro (menos o último)
- Um caractere especial de quebra de linha é impresso após o último argumento

- Exemplo

```
def soma(a, b):  
    print('O resultado de', a, '+', b, 'é')  
    print(a + b)
```

```
>>> soma(2, 3)  
O resultado de 2 + 3 é  
5  
>>>
```

## Entrada e saída padrão

- A função `print` em geral é utilizada em conjunto com a função `string.format`

```
def soma(a, b):  
    print('O resultado de {} + {} é'.format(a, b))  
    print(a + b)
```

```
>>> soma(2, 3)  
O resultado de 2 + 3 é  
5  
>>>
```

- `string.format` faz a formatação da string e `print` exibe a string

- A função `string.format` devolve a string de formatação substituindo cada item `{}` por um argumento

```
>>> 'O resultado de {} + {} é'.format(1/3, 2/3)
```

```
'O resultado de 0.3333333333333333 + 0.6666666666666666 é'
```

- Entre os símbolos { e } pode ser especificado a formatação do item

```
>>> 'O resultado de {:.2f} + {:.4f} é'.format(1/3, 2/3)  
O resultado de 0.33 + 0.6667 é'
```

- Existem muitas opções para a formatação, veja os detalhes em <https://docs.python.org/3/library/string.html#formatspec>

Vamos escrever um programa que lê 3 notas digitas pelo usuário, calcula a média das notas, e exibe o resultado.

- Primeiro criamos a função que calcula a média usando a receita de projeto
- Em seguida criamos uma função principal que faz a leitura dos dados, chama a função para calcular a média e depois exibe o resultado
- O trecho de código que faz a leitura (escrita) dos dados também pode ser colocado em funções separadas

- Arquivo `media.py`

```
def main():  
    print('Este programa calcula a média de 3 notas')  
    a = float(input('Nota 1: '))  
    b = float(input('Nota 2: '))  
    c = float(input('Nota 3: '))  
    m = media(a, b, c)  
    print('A média é {0:.2f}'.format(m))  
  
def media(a, b, c):  
    return (a + b + c) / 3
```

# Execução de programas no terminal

## Execução de programas no terminal

- A função `main` do programa anterior pode ser executada na janela de interações do editor `mu`, mas como permitir que um usuário sem o editor `mu` execute a função?
- O programas Python podem ser executados no terminal

```
$ python3 arquivo.py
```

## Execução de programas no terminal

- Seja `media.py` o arquivo contendo a função `main` e `media` que definimos anteriormente, se executarmos o programa no terminal

```
$ python3 media.py
```

- Ele não executará nenhum função...

- Precisamos indicar que função deve ser executada quando o programa é executado. Fazemos isto escrevendo o seguinte trecho de código no final do arquivo `media.py`

```
if __name__ == "__main__":  
    main()
```

# Execução de programas no terminal

- Executando o programa

```
$ python3 media.py
```

```
Este programa calcula a média de 3 notas
```

```
Nota 1: 6
```

```
Nota 2: 7
```

```
Nota 3: 9
```

```
A média é 7.33
```

# Execução de programas no terminal

- Também podemos executar os testes do nosso programa no terminal

```
$ python3 -m doctest -v media.py
Trying:
media(1.0, 2.0, 3.0)
Expecting:
    2.0
ok
Trying:
    media(3.0, 5.0, 10.0)
Expecting:
    6.0
ok
2 items had no tests:
    media
    media.main
1 items passed all tests:
    2 tests in media.media
2 tests in 3 items.
2 passed and 0 failed.
Test passed.
```

# Arquivos

- Muitos programas leem os dados de entrada de arquivos e/ou escrevem os dados de saída em arquivos
- Veremos como ler e escrever arquivos texto usando o Python

# Arquivos

- Antes de ser lido, um arquivo precisa ser aberto (o arquivo é fechado automaticamente pela instrução `with`)
- Após aberto, um arquivo texto pode ser lido linha por linha usando um `for`

```
def le_arquivo(arquivo):  
    linhas = []  
    with open(arquivo) as f:  
        # linha conterá o símbolo de quebra de linha  
        # se presente na entrada  
        for linha in f:  
            linhas.append(linha)  
    return linhas
```

- Considerando um arquivo `exemplo.txt` com as linhas

```
Este é um arquivo de exemplo
```

```
Tem números como
```

```
23 50
```

```
100
```

```
e outras coisas.
```

- A função `le_arquivo('exemplo.txt')` produz

```
>>> le_arquivo('exemplo.txt')
```

```
['Este é um arquivo de exemplo\n',
```

```
 'Tem números como\n',
```

```
 '23 50\n', '100\n',
```

```
 'e outras coisas.\n']
```

- Para escrever em um arquivo, ele precisa ser aberto em modo escrita (argumento 'w')
- Após aberto em modo escrita, usamos a função write para escrever no arquivo

```
def escreve_arquivo(arquivo, linhas):  
    with open(arquivo, 'w') as f:  
        for linha in linhas:  
            f.write(linha)  
            # grava o símbolo de quebra de linha  
            f.write('\n')  
    return linhas
```

- Após a execução de

```
>>> escreve_arquivo('exemplo.txt',  
                    ['Casa',  
                    '23',  
                    'outra coisa'])
```

- O conteúdo do arquivo exemplo.txt será

Casa

23

outra coisa

# Parâmetros na linha de comando

## Parâmetros na linha de comando

- É possível passar argumentos na linha de comando para um programa escrito em Python

```
$ python3 arquivo.py parametro1 parametro2 ...
```

## Parâmetros na linha de comando

- Vamos escrever um programa que recebe como parâmetro dois número e exhibe na tela a soma dos dois números
- Queremos que o nosso programa funcione da seguinte maneira

```
$ python3 soma.py 10 5
```

```
15
```

```
$ python3 soma.py 1 -1
```

```
0
```

## Parâmetros na linha de comando

- Arquivo soma.py

```
import sys
```

```
def main():
```

```
    a = int(sys.argv[1])
```

```
    b = int(sys.argv[2])
```

```
    print(a + b)
```

```
if __name__ == "__main__":
```

```
    main()
```

- Os argumentos são armazenados como strings na lista `sys.argv` a partir da posição 1
- O valor na posição 0 de `sys.argv` é o nome do arquivo `.py`

## Parâmetros na linha de comando

- Note que se não especificarmos os parâmetros os especificarmos parâmetros inválidos, o programa vai gerar um erro

```
$ python3 soma.py
```

```
Traceback (most recent call last):
```

```
  File "src/10/soma.py", line 9, in <module>  
    main()
```

```
  File "src/10/soma.py", line 4, in main  
    a = int(sys.argv[1])
```

```
IndexError: list index out of range
```

- Para facilitar o uso do programa, é necessário validar os parâmetros e informar erros mais claros

# Atividades

Escreva um programa que receba como argumento o nome de um arquivo de pontos cartesianos e mostre os dois pontos mais distantes e a distância entre eles. O arquivo deve conter um ponto por linha (dois número separados por espaço).

Veja o arquivo `pontos-mais-distantes.py`

Execução do programa

```
$ python3 pontos-mais-distantes.py pontos.txt
```