# Aula 1 - PARI

*Sumário*



*Pressupostos para a realização dos exercícios*

- Ter o Linux instalado (ubuntu 18.04 **altamente recomendado**).
- Ter o acesso de rede configurado (*wireless*).
    - Consultar as instruções do site dos sTIC.

# Instalação do Ubuntu

Existem diveras formas de usar o Ubuntu para quem tem outros sistemas operativos (Windows, MacOS). As mais interessantes são:

1. Uso do live ubuntu: Try before you install
2. Instalar uma máquina virtual (`virtualbox` ou outra) e instalar o Linux na máquina virtual.
3. Instalar o Linux em dual-boot com o Windows (Esta é a forma recomendada. As outras formas serão pouco adequadas em breve).

As duas primeiras soluções não interferem no disco nem no sistema operativo existente, mas são mais limitadas em termos de funcionalidades e desempenho. No caso da primeira, todo o trabalho que for feito se perde no fim da sessão se não for copiado para outro local. No caso da máquina virtual, vai ser preciso espaço em disco no ambinente Windows (ou MacOs) para criar a "imagem" do disco onde correrá o Linux em máquina virtual. É uma solução intermédia que funciona relativamente bem, mas como opera sobre o sistema operativo nativo, pode ter limitações de desempenho e ficará dependente da atividade desse sistema operativo (como as atualizações no Windows).

A terceira solução (dual-boot com o sistema operativo nativo) é a mais poderosa porque cada sistema operativo fica no seu próprio espaço e correm separadamente. Porém, é preciso repartir o disco que estaria todo atribuído ao sistema operativo nativo. O Linux oferece esta possibilidade durante a instalação e em geral o processo corre bem, mas há sempre o risco de perda de informação. Por isso, recomenda-se guardar toda a informação importante desenvolvida no sistema operativo original antes de fazer esta forma de instalação.

Mais informações podem ser obtidas por exemplo nos seguintes endereços:

- https://tutorials.ubuntu.com/
- https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop

# Notas iniciais sobre os ficheiros de apoio às aulas

*Transformar os ficheiros de Sumários e Exercícios de ASCIIDOC para HTML*

Estes ficheiros de Sumários são legíveis no formato ASCII, mas podem ser convertidos em HTML para uma melhor apresentação usando o comando `asciidoc`. Este comando pode ser instalado no Ubuntu com `sudo apt install asciidoc` bastando depois executar na linha de comando `asciidoc SumarioA1.adoc`, ou seja, num terminal (que é uma janela onde se escrevem comandos manualmente --- essa janela pode ser lançada a partir da interface gráfica do linux a partir da zona das aplicações). NOTA: Antes de fazer instalações pode ser necessário atualizar as fontes dos repositórios; isso faz-se com `sudo apt update`. Para fazer o `syntax highlight` do código de programação incluído neste ficheiro o programa recorre a comandos externos. O de defeito é o `source-highlight` e que se se deve instalar previamente, por exemplo, com o comando `sudo apt install source-highlight`.

Como exercício, sugere-se tentar criar o ficheiro em formato HTML (`SumarioA1.html`) a partir do ficheiro em formato asciidoc (`SumarioA1.adoc`).

*Transformar os ficheiros de Sumários e Exercícios de ASCIIDOC para PDF*

Os documentos em formato ASCIIDOC também podem ser convertidos em PDF recorrendo ao comando `a2x`, que é uma forma elaborada de usar o `asciidoc` e outras packages auxiliares como o `dblatex` (por defeito) ou o `FOP` para gerar outros formatos (dos quais o PDF é o de defeito). Para serem usados, estes comandos também devem ser instalados: `sudo apt install dblatex fop`. Depois, nesse caso, bastará fazer `a2x SumarioA1.adoc` ou `a2x --fop SumarioA1.adoc` respetivamente. Qualquer destes comandos tem opções na linha de comando que os tornam configuráveis. O `dblatex` pode ter problemas com ficheiros com caracteres especiais (por exemplo UTF-8) e isso tem de ser prevenido para executar sem erros. Por vezes isso torna-se complexo e a opção `a2x --fop SumarioA1.adoc` acaba por ser a mais cómoda.

*Nomes das linguagens para o syntax highlight e o formato PDF*

Quando se gera o documento em PDF, pode haver conflitos de nomes nas linguagens para o *syntax highlight*. Um desses casos é a linguagem de `makefile` que é reconhecida pelo `source-highlight` mas não pelo `highlight` nem pelo `pygments` que ainda por cima disso espera `c` e não `C` para código fonte em linguagem C. Esta questão alarga-se quando se usa o comando `a2x` para gerar PDF porque ele recorre ao `dblatex` que tem o seu próprio `syntax highlight` para gerar PDF e não reconhece a linguagem `makefile` mas sim `make` (embora pobre!). Nesse caso, e se se pretende criar o PDF, ou se usa a variante `a2x --fop`, que não faz qualquer `syntax highlight`, ou então usa-se o `a2x` de defeito (`dblatex`), e altera-se o tipo de linguagem *source* de `makefile` para `make`; porém aí é o `asciidoc` que não funcionará bem se não se usar o `source-highlighter`. A questão pode-se tornar confusa e há, portanto, uma questão de compromissos. O `syntax highlight` gerado pelo `source-highlight` é em geral mais rico que os outros, portanto recomenda-se. Se for para gerar PDF, ou se abdica do `syntax highlight` (usando a2x --fop) ou se ajusta as linguagens (mormente de `makefile` para `make`) e usa-se o `dblatex`, que é o processador de defeito do `a2x`.

# Parte 1 - Apresentação e generalidades

Ver o documento `0-PARI2019-2020-Apresentação`

# Parte 2 - Introdução ao Linux e a Shell

Ver o documento `1-Linux-Breve Introdução`

# Parte 3 - Criação do ambiente e instalação de ferramentas básicas

## Metodologia

Para melhor se desenvolver o trabalho nas aulas, deve-se seguir uma metodologia de organização de ficheiros em diretórios por aulas e por exercícios.

Dentro de cada aula, em especial nas primeiras, é também recomendado criar uma subpasta para cada exercício `Ex1`, `Ex2`, etc. Em certas aulas, ou aulas mais avançadas, os diversos exercícios serão feitos por acréscimo sucessivo sobre o código base dos exercícios anteriores; nessa altura serão dadas as instruções nesse sentido.

Os guiões para as aulas estarão a ser continuamente atualizados em:

https://github.com/miguelriemoliveira/pari_20-21

Recomenda-se que, sempre que possível, usem a versão online.

## Editor

A ferramenta principal para criar e modificar ficheiros é o editor, muitas vezes integrado num ambiente de desenvolvimento (IDE). Há inúmeras opções desde simples editores (`gedit`, `kate`, `kwrite`, etc.) até ambientes de desenvolvimento muito sofisticados (`codeblocks`, `eclipse`, `vscode`, etc.).

Além das propriedades fundamentais dos editores, hoje em dia são excelentes *add-ons* a "automated completion" (preenchimento automático de palavras e estruturas) , o "syntax highlight" (realce da sintaxe da linguagem), o "intellissense" (apresentação de todas as opções de preenchimento automático de campos e estruturas em variáveis, funções, etc.), ou a inserção automática de fragmentos de código padrão ("code snippets").

O editor com mais tradição por excelência é o "vim" (ou "vi" improuved) mas a sua utilização eficaz pode requerer anos de prática continuada e permite todas as facilidades indicadas acima, mas a sua configuração, por ser praticamente ilimitada, pode-se tornar complexa e, por isso, contraproducente em utilizadores iniciados.

Recomenda-se como IDE o pycharm. Pode-se utilizar a licença profissional ou a licença base. Os estudantes podem solicitar acesso à licença profissional.

Outra alternativa uma solução disponibilizada pela Microsoft para muitas plataformas, incluindo linux. Trata-se do *Visual Studio Code* ou *vscode*. Pode ser instalado diretamente do gestor de aplicações do Ubuntu (*Ubuntu Software*) ou por outras vias (https://askubuntu.com/questions/616075/how-do-i-install-visual-studio-code).

# Parte 4 - Primeiros exercícios de programação em Python

## Exercício 1

Desenvolver um programa que imprima no terminal a frase "Hello World". Editar o ficheiro `hello.py` com o editor escolhido (`gedit`, `kate`, etc.)

*hello.py*

```
def main():
    print("Hello World!")

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

executar o programa com o seguinte comando:

```
python ex1.py
```

# Exercício 2

Criar um programa designado `primos` que imprime no ecran números primos, um por linha, até um certo limite. Usar uma função auxiliar `isPrime()` que aceita um inteiro `n` e retorna 1 ou 0 conforme `n` for primo ou não.

*primo.py*

```
maximum_number = 50


def isPrime(value):
    # <Fill the blank>

def main():
    print("Starting to compute prime numbers up to " + str(maximum_number))

    for i in range(0, maximum_number):
        if isPrime(i):
            print('Number ' + str(i) + ' is prime.')
        else:
            print('Number ' + str(i) + ' is not prime.')

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

Com a ajuda do programa, calcular quantos números primos inferiores a 10000 têm o algarismo 3.

```
python primos.py | grep "3" | wc -l
```

A resposta deve ser 561

# Exercício 3

Estender o exercício 2 de modo a:

1. Imprimir todos os divisores calculados para os números não primos;
2. Usar o package colorama para imprimir os números primos a verde;
3. Usar *shebang line* para simplificar execução do script;

# Exercício 4

Calcular números perfeitos (aqueles cuja soma dos divisores igualam o número) como por exemplo 6 = 3 + 2 + 1. Além do `main()` criar a função `isPerfect()`, que indica se o número é perfeito.

*perfeitos.py*

```
maximum_number = 100

def isPerfect(value):
    # <Fill the blanks>
    return False

def main():
    print("Starting to compute perfect numbers up to " + str(maximum_number))

    for i in range(0, maximum_number):
        if isPerfect(i):
            print('Number ' + str(i) + ' is perfect.')


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