**Laboratório prático – Módulo 1**

**Parte 1: Testes empíricos**

Criar *arrays* usando a biblioteca numérica do Python [*NumPy*](https://numpy.org/) e anotar tempos de execução.

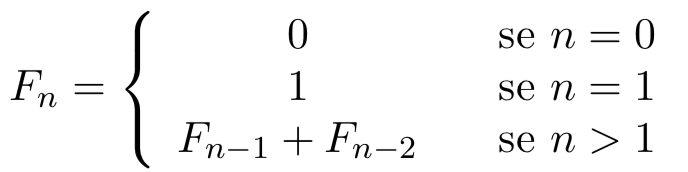
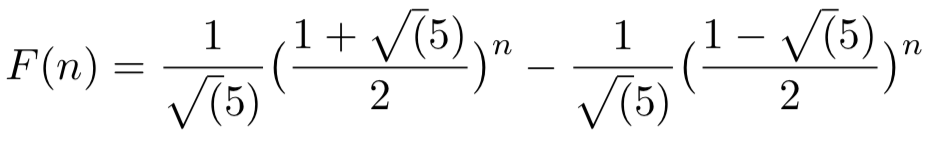
A biblioteca [*numpy*](https://numpy.org/doc/stable/user/absolute_beginners.html) oferece-nos a possibilidade de criar *arrays* com operadores numéricos otimizados. Note que, contrariamente a uma *list*, um *array* pode apenas ter um tipo de dados.

No que segue, pode usar o notebook que encontra no Moodle e que, tal como o exemplo disponibilizado para a pré-aula, introduz a estrutura de dados *numpy array*

1. Considere as seguintes funções:

Usando os módulos *plot* da biblioteca[*matplotlib*](https://matplotlib.org/), gere gráficos para as funções acima. Faça x tomar valores inteiros no intervalo.

Após observação dos gráficos, que conclusões pode estabelecer?

1. Implemente os algoritmos para multiplicação de inteiros na aula teórica-prática (semana 1) e replique a informação aí indicada (ver slides da aula ou, se quiser, pode usar os materiais auxiliares dessa semana).
2. Considere a sucessão de Fibonacci:
   * 1. Implemente três algoritmos para calcular e devolver o n-ésimo número de Fibonacci usando:
        + uma implementação recursiva - fiboR
        + uma implementação de programação dinâmica - fiboP
        + usando a fórmula:
3. Execute as suas 3 implementações para calcular F(n), com n = 10, 15, ..., 40, **anotando o tempo** que demora cada execução (não esqueça de repetir 5 ou mais vezes para cada valor de n e anotar (apenas) os tempos médios[[1]](#footnote-1)). Nota: se alguma versão de implementação demorar “demasiado” tempo, termine essa execução e anote como INF.
4. Compare os tempos médios de execução entre as 3 implementações para cada valor de n. Que conclusões pode estabelecer?
5. Compare os valores finais de resposta para o número de Fibonacci devolvidos pelas 3 implementações para cada valor de n. Que conclusões pode estabelecer?
6. Indique, justificando, quais as ordens de complexidade *big-O* de cada um dos algoritmos implementados.
7. (Exercício pré-aula para preparação das aulas TP da semana 2.)
   1. Crie um segmento de código para gerar uma coleção aleatória de números inteiros entre 1 e 10000 e guardar esses valores numa coleção do tipo:
      * + - estrutura *list*;
          - *array numpy*;
          - estrutura *set*.
   2. Implemente código Python para definir **três** funções para efetuar pesquisa de um inteiro numa dada coleção: uma função usando o operador para coleções “in” do Python, outra função usando um algoritmo de pesquisa sequencial criada por si e uma outra usando pesquisa binária. (Nota: tenha em atenção a especificação de entrada do algoritmo para a pesquisa binária).
   3. Para cada função e para cada tipo de coleção (e usando exatamente a mesma coleção, gerada aleatoriamente antes das invocações às funções de pesquisa nos respetivos testes), cronometre o tempo de execução de cada pesquisa de um dado elemento, repetindo o mesmo 35 vezes[[2]](#footnote-2) e tirando a média. Este teste deve ser efetuado usando para a pesquisa:
8. um elemento que não existe na coleção: 10100;
9. um elemento que não existe na coleção: -1;
10. um elemento da lista (escolha aleatoriamente um elemento da coleção após a sua geração e use o mesmo elemento em todos os testes).
11. Faça o *plot* dos tempos de (c.i) e (c.ii). Que conclusões tira? (Estabeleça ligações entre os tempos médios e as ordens de complexidade conhecidas para algoritmos de pesquisa linear e pesquisa binária).
12. Use agora uma inicialização sequencial da sua coleção de inteiros e faça novas cronometragens. Explique se encontrou diferenças e explique porquê.

1. Faça uma outra experiência prévia: execute 5 ou mais execuções para um valor de n > 10, mas guardando o tempo individual da execução e verificando, no final, quais foram esses tempos individuais. São todos iguais? Este comportamento justifica o pedido de anotar apenas o valor médio dos tempos de repetição para cada n? [↑](#footnote-ref-1)
2. O número **35** não é ao acaso: prova-se que, para fazer testes significativos, ou seja, que sejam passíveis de generalização para uma utilização em ambiente real, **o número de repetições para avaliar um valor médio deve ser superior a 32**. Habitualmente, devido a, em geral, as execuções poderem ser de problemas de resolução computacional demorada, a quantidade de repetições mais vista é 10 ou 12. [↑](#footnote-ref-2)