# Princípios de Programação

# Projeto 1

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Engenharia Informática

2023/2024

O objetivo do primeiro projeto é que os alunos desenvolvam familiaridade com a sintaxe da linguagem Haskell, nomeadamente a escrita de funções simples e a definição de listas em compreensão. Pretende-se que após este projeto, os alunos sejam capazes de compreender e utilizar padrões, guardas, expressões com `where` e `let`, e ainda listas em compreensão com vários geradores e filtros.

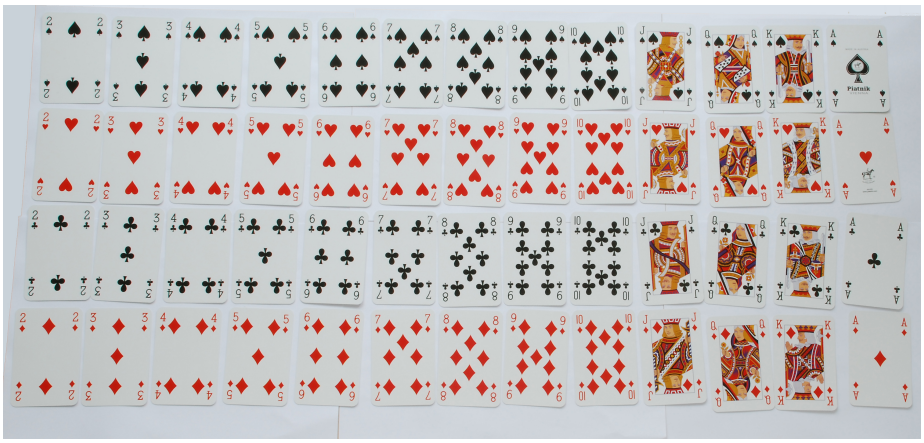

Figura 1: Baralho de 52 cartas. Fonte: `https://commons.wikimedia.org/w/index.php?curid=7104281`

Para tal, vamos desenvolver duas pequenas aplicações para jogos de cartas. Um baralho normal é composto por 52 cartas. Cada carta é identificada pelo seu valor (13 possibilidades) e pelo seu naipe (4 possibilidades).
Vamos representar um valor como um caracter de entre:

- `'A'` : ás (em inglês, *ace*)
- `'2'` a `'9'` : dois a nove
- `'T'` : dez (em inglês, *ten*)
- `'J'` : valete (em inglês, *jack*)
- `'Q'` : dama (em inglês, *queen*)
- `'K'` : rei (em inglês, *king*)

Também vamos representar um naipe como um caracter, de entre:

- `'S'` : espadas (em inglês, *spades*)
- `'H'` : copas (em inglês, *hearts*)
- `'D'` : ouros (em inglês, *diamonds*)
- `'C'` : paus (em inglês, *clubs*)

Vamos representar uma carta como uma **`String`** de tamanho dois. O primeiro caracter corresponde ao valor da carta e o segundo caracter corresponde ao seu naipe. Exemplos:

- `"AS"` : ás de espadas
- `"4D"` : quatro de ouros
- `"TC"` : dez de paus
- `"KH"` : rei de copas

Assim sendo, um baralho é composto pelas seguintes cartas:

```
"AS","2S","3S","4S","5S","6S","7S","8S","9S","TS","JS",
"QS","KS","AH","2H","3H","4H","5H","6H","7H","8H","9H",
"TH","JH","QH","KH","AD","2D","3D","4D","5D","6D","7D",
"8D","9D","TD","JD","QD","KD","AC","2C","3C","4C","5C",
"6C","7C","8C","9C","TC","JC","QC","KC"
```

**A. Baralho** Defina uma lista `baralho` com as 52 cartas do baralho. A sua definição deve usar uma lista em compreensão, e não uma mera listagem das cartas do baralho.
Exemplos de utilização

```
> length baralho
52
> elem "AD" baralho
True
> elem "XY" baralho
False
```

**B. Blackjack** O Blackjack é um jogo de apostas com cartas. Neste jogo, cada carta vale um certo número de pontos, e o objetivo é chegar o mais próximo possível dos 21 pontos sem ultrapassar os 21 pontos. Para os efeitos deste projeto, vamos usar uma versão simplificada das regras oficiais. Cada carta vale os seguintes pontos:

- ás: vale 1 ou 11 pontos, à escolha do jogador
- dois a dez: vale 2 a 10 pontos, respetivamente
- figuras (valete, dama, rei): vale 10 pontos

No início, o jogador recebe duas cartas. Vamos utilizar um par (tuplo de tamanho 2) para representar a mão inicial. Exemplos:

- `("2S","JH")` : dois e valete, vale 12 pontos
- `("4H","4C")` : par de quatros, vale 8 pontos
- `("5C","AD")` : cinco e ás, vale 6 ou 16 pontos, à escolha do jogador
- `("AD","JD")` : ás e valete, vale 11 ou 21 pontos, à escolha do jogador.

As melhores mãos iniciais são constituídas por um ás e um dez ou figura, pois permitem atingir a pontuação perfeita de 21 pontos.
Defina uma função `combinacoesBlackjack` que, dado um número de pontos, devolva a lista de todas as mãos iniciais de Blackjack (pares de cartas) com esse número de pontos. Por exemplo, as únicas mãos iniciais que valem 3 pontos são constituídas por dois e ás.

```
> combinacoesBlackjack 3
[("2S","AS"), ("2S","AH"), ("2S","AD"), ("2S","AC"),
 ("2H","AS"), ("2H","AH"), ("2H","AD"), ("2H","AC"),
 ("2D","AS"), ("2D","AH"), ("2D","AD"), ("2D","AC"),
 ("2C","AS"), ("2C","AH"), ("2C","AD"), ("2C","AC")]
> length (combinacoesBlackjack 3)
16
```

A sua implementação deve assumir o seguinte.

- Estamos a jogar só com um baralho, pelo que as cartas de um par devem ser diferentes (não é possível ter, por exemplo, dois valetes de copas)
- A ordem das cartas numa mão não interessa. Por exemplo, as mãos `("AS","2S")` e `("2S","AS")` devem ser consideradas idênticas. A solução implementada não pode incluir duas mãos idênticas.
- A ordem pela qual as cartas aparecem numa mão, ou pela qual as mãos aparecem na lista resultado, não é relevante para a avaliação. No exemplo acima (`combinacoesBlackjack 3`), é permitido que a sua solução devolva uma lista ligeiramente diferente, em que as mãos estejam numa outra ordem, ou em que as cartas de cada mão estejam numa outra ordem.

**C. Poker** O Poker é um outro jogo de cartas. Neste jogo, uma mão é composta por cinco cartas. Vamos representar cada mão por uma lista de cinco cartas. Há várias combinações de mãos possíveis. Uma das combinações mais valiosas chama-se *Full House*, que corresponde a um trio (três cartas do mesmo valor) mais um par (duas cartas do mesmo valor). Por exemplo, `["TS","TH","TD","QH","QC"]` corresponde a um *full house* de 10 e damas.
Defina uma lista `fullHouses` de todas as mãos de Poker que correspondam a *full houses*. Examplos de utilização:

```
> take 5 fullHouses
[["AD","AH","AS","2H","2S"], ["AD","AH","AS","3H","3S"],
 ["AD","AH","AS","4H","4S"], ["AD","AH","AS","5H","5S"],
 ["AD","AH","AS","6H","6S"]]
> length fullHouses
3744
```

A sua implementação deve assumir o seguinte, de forma semelhante ao exercício do Blackjack.

- Estamos a jogar só com um baralho, pelo que as cartas de uma mão devem ser diferentes.
- A solução implementada não pode incluir duas mãos idênticas (com as mesmas cartas mas numa ordem diferente).
- A ordem pela qual as cartas aparecem numa mão, ou pela qual as mãos aparecem na lista resultado, não é relevante para a avaliação. É permitido que a sua solução seja ligeiramente diferente da solução acima, em que as mãos estejam numa outra ordem, ou em que as cartas de cada mão estejam numa outra ordem.

**Notas**

1. Deverá submeter um ficheiro com o nome `p1_XXXXX.hs`, onde `XXXXX` é o seu número de aluno.
2. Os trabalhos serão avaliados automaticamente. Respeite os nomes e os tipos das funções e listas enunciadas acima.
3. Cada função (ou expressão) que escrever deverá vir sempre acompanhada de uma assinatura. Isto é válido para as funções ou expressões enunciadas acima bem como para outras funções ou expressões ajudantes que decidir implementar.
4. Para resolver estes problemas deve utilizar *apenas* a matéria dos três primeiros capítulos do livro (de "Starting Out" até "Syntax in Functions"). Pode usar qualquer função constante no `Prelude`.
5. **Não pode em caso algum definir funções por recursão.**

6. Lembre-se que as boas práticas de programação Haskell apontam para a utilização de várias funções simples em lugar de uma função única mas complicada.

**Entrega.** Este é um trabalho de resolução individual. Os trabalhos devem ser entregues no Moodle até às 23:55 do dia 16 de outubro de 2023.

**Plágio.** A nível académico, alunos detetados em situação de fraude ou plágio (plagiadores e plagiados) em alguma prova ficam reprovados à disciplina. Serão ainda alvo de processo disciplinar, ficando registado no processo de aluno, podendo conduzir à suspensão letiva, expulsão da universidade e/ou denúncia no Ministério Público.
Qualquer situação em que um aluno submete material que não é da sua autoria é considerada fraude ou plágio. Isto inclui material da autoria de colegas, de terceiros, de fontes online não identificadas ou por inteligência artificial generativa (ex: ChatGPT). Tais ferramentas são portanto proibidas na realização dos projetos.
Todos os trabalhos submetidos são submetidos a uma ferramenta de verificação de semelhanças de software. Aqueles em que o sistema assinalar um elevado grau de semelhança são posteriormente analisados manualmente pelos docentes.