

Programação Funcional



Capítulo 1 Paradigmas de Programação

José Romildo Malaquias

Departamento de Computação
Universidade Federal de Ouro Preto

2018.1

- 1 Paradigmas de programação
- 2 Programação funcional
- 3 A crise do software
- 4 Algumas características de Haskell
- 5 Antecedentes históricos
- 6 Algumas empresas que usam Haskell
- 7 Curso online
- 8 Programando com Funções
- 9 Experimentando Haskell

- 1 Paradigmas de programação
- 2 Programação funcional
- 3 A crise do software
- 4 Algumas características de Haskell
- 5 Antecedentes históricos
- 6 Algumas empresas que usam Haskell
- 7 Curso online
- 8 Programando com Funções
- 9 Experimentando Haskell

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - **Smalltalk**, **Eiffel** e **Java** suportam o paradigma orientado a objetos.
 - **Haskell** e **Clean** suportam o paradigma funcional.
 - **OCaml**, **LISP**, **Scala**, **Perl**, **Python** e **C++** suportam múltiplos paradigmas.

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - **Smalltalk**, **Eiffel** e **Java** suportam o paradigma orientado a objetos.
 - **Haskell** e **Clean** suportam o paradigma funcional.
 - **OCaml**, **LISP**, **Scala**, **Perl**, **Python** e **C++** suportam múltiplos paradigmas.

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - **Smalltalk**, **Eiffel** e **Java** suportam o paradigma orientado a objetos.
 - **Haskell** e **Clean** suportam o paradigma funcional.
 - **OCaml**, **LISP**, **Scala**, **Perl**, **Python** e **C++** suportam múltiplos paradigmas.

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - **Smalltalk**, **Eiffel** e **Java** suportam o paradigma orientado a objetos.
 - **Haskell** e **Clean** suportam o paradigma funcional.
 - **OCaml**, **LISP**, **Scala**, **Perl**, **Python** e **C++** suportam múltiplos paradigmas.

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - **Smalltalk**, **Eiffel** e **Java** suportam o paradigma orientado a objetos.
 - **Haskell** e **Clean** suportam o paradigma funcional.
 - **OCaml**, **LISP**, **Scala**, **Perl**, **Python** e **C++** suportam múltiplos paradigmas.

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - Smalltalk, Eiffel e Java suportam o paradigma orientado a objetos.
 - Haskell e Clean suportam o paradigma funcional.
 - OCaml, LISP, Scala, Perl, Python e C++ suportam múltiplos paradigmas.

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - Smalltalk, Eiffel e Java suportam o paradigma orientado a objetos.
 - Haskell e Clean suportam o paradigma funcional.
 - OCaml, LISP, Scala, Perl, Python e C++ suportam múltiplos paradigmas.

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - **Smalltalk**, **Eiffel** e **Java** suportam o paradigma orientado a objetos.
 - **Haskell** e **Clean** suportam o paradigma funcional.
 - **OCaml**, **LISP**, **Scala**, **Perl**, **Python** e **C++** suportam múltiplos paradigmas.

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - **Smalltalk**, **Eiffel** e **Java** suportam o paradigma orientado a objetos.
 - **Haskell** e **Clean** suportam o paradigma funcional.
 - **OCaml**, **LISP**, **Scala**, **Perl**, **Python** e **C++** suportam múltiplos paradigmas.

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - **Smalltalk**, **Eiffel** e **Java** suportam o paradigma orientado a objetos.
 - **Haskell** e **Clean** suportam o paradigma funcional.
 - **OCaml**, **LISP**, **Scala**, **Perl**, **Python** e **C++** suportam múltiplos paradigmas.

Paradigmas de programação

- Um **paradigma de programação** determina a visão que o programador possui sobre a **estruturação** e a **execução** do programa.
- Por exemplo:
 - Em **programação orientada a objetos**, programadores podem abstrair um programa como uma *coleção de objetos* que interagem entre si.
 - Em **programação lógica** os programadores abstraem o programa como um *conjunto de predicados* que estabelecem relações entre objetos (axiomas), e uma *meta* (teorema) a ser provada usando os predicados.
 - Em **programação funcional** os programas são expressões a serem avaliadas, e são organizados usando funções.
- Diferentes linguagens de programação suportam diferentes paradigmas de programação.
- Algumas linguagens foram desenvolvidas para suportar um paradigma específico.
- Por exemplo:
 - **Smalltalk**, **Eiffel** e **Java** suportam o paradigma orientado a objetos.
 - **Haskell** e **Clean** suportam o paradigma funcional.
 - **OCaml**, **LISP**, **Scala**, **Perl**, **Python** e **C++** suportam múltiplos paradigmas.

Técnicas e paradigmas de programação

- Geralmente os paradigmas de programação são diferenciados pelas **técnicas de programação que permitem ou proíbem**.
- Por exemplo, a **programação estruturada** não permite o uso de *goto*.
- Esse é um dos motivos pelos quais novos paradigmas são considerados mais rígidos que estilos tradicionais.
- Apesar disso, evitar certos tipos de técnicas pode facilitar a prova de correção de um sistema, podendo até mesmo facilitar o desenvolvimento de algoritmos.
- O relacionamento entre paradigmas de programação e linguagens de programação pode ser complexo pelo fato de linguagens de programação poderem **suportar mais de um paradigma**.

Técnicas e paradigmas de programação

- Geralmente os paradigmas de programação são diferenciados pelas **técnicas de programação que permitem ou proíbem**.
- Por exemplo, a **programação estruturada** não permite o uso de *goto*.
- Esse é um dos motivos pelos quais novos paradigmas são considerados mais rígidos que estilos tradicionais.
- Apesar disso, evitar certos tipos de técnicas pode facilitar a prova de correção de um sistema, podendo até mesmo facilitar o desenvolvimento de algoritmos.
- O relacionamento entre paradigmas de programação e linguagens de programação pode ser complexo pelo fato de linguagens de programação poderem **suportar mais de um paradigma**.

Técnicas e paradigmas de programação

- Geralmente os paradigmas de programação são diferenciados pelas **técnicas de programação que permitem ou proíbem**.
- Por exemplo, a **programação estruturada** não permite o uso de *goto*.
- Esse é um dos motivos pelos quais novos paradigmas são considerados mais rígidos que estilos tradicionais.
- Apesar disso, evitar certos tipos de técnicas pode facilitar a prova de correção de um sistema, podendo até mesmo facilitar o desenvolvimento de algoritmos.
- O relacionamento entre paradigmas de programação e linguagens de programação pode ser complexo pelo fato de linguagens de programação poderem **suportar mais de um paradigma**.

Técnicas e paradigmas de programação

- Geralmente os paradigmas de programação são diferenciados pelas **técnicas de programação que permitem ou proíbem**.
- Por exemplo, a **programação estruturada** não permite o uso de *goto*.
- Esse é um dos motivos pelos quais novos paradigmas são considerados mais rígidos que estilos tradicionais.
- Apesar disso, evitar certos tipos de técnicas pode facilitar a prova de correção de um sistema, podendo até mesmo facilitar o desenvolvimento de algoritmos.
- O relacionamento entre paradigmas de programação e linguagens de programação pode ser complexo pelo fato de linguagens de programação poderem **suportar mais de um paradigma**.

Técnicas e paradigmas de programação

- Geralmente os paradigmas de programação são diferenciados pelas **técnicas de programação que permitem ou proíbem**.
- Por exemplo, a **programação estruturada** não permite o uso de *goto*.
- Esse é um dos motivos pelos quais novos paradigmas são considerados mais rígidos que estilos tradicionais.
- Apesar disso, evitar certos tipos de técnicas pode facilitar a prova de correção de um sistema, podendo até mesmo facilitar o desenvolvimento de algoritmos.
- O relacionamento entre paradigmas de programação e linguagens de programação pode ser complexo pelo fato de linguagens de programação poderem **suportar mais de um paradigma**.

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando *como resolver* um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não *como* seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas
 - **funcional**: Haskell, OCaml, Lisp, etc.
 - **declarativo**: Prolog, etc.

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando **como resolver** um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não **como** seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas
 - **funcional**: Haskell, OCaml, Elm, etc
 - **base de dados**: Prolog, etc

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando **como resolver** um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: *faça isso, depois isso, depois aquilo...*
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não *como* seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas
 - **funcional**: Haskell, OCaml, Erlang, etc
 - **base de dados**: Prolog, etc

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando **como resolver** um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não **como** seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando **como resolver** um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não **como** seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando *como resolver* um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve *o que* o programa faz e não *como* seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando **como resolver** um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não **como** seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando *como resolver* um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não *como* seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas
 - **funcional**: Haskell, OCaml, LISP, etc, e
 - **lógico**: Prolog, etc.

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando *como resolver* um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não *como* seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas
 - **funcional**: Haskell, OCaml, LISP, etc, e
 - **lógico**: Prolog, etc.

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando *como resolver* um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não *como* seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas
 - **funcional**: Haskell, OCaml, LISP, etc, e
 - **lógico**: Prolog, etc.

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando *como resolver* um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: *faça isso, depois isso, depois aquilo...*
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não *como* seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas
 - **funcional**: Haskell, OCaml, LISP, etc, e
 - **lógico**: Prolog, etc.

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando *como resolver* um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não *como* seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas
 - **funcional**: Haskell, OCaml, LISP, etc, e
 - **lógico**: Prolog, etc.

Categorias: programação imperativa e declarativa

Programação imperativa

- Descreve a computação como ações, enunciados ou comandos que **mudam o estado** (variáveis) de um programa, enfatizando *como resolver* um problema.
- Um programa imperativo é uma **sequência de comandos** para o computador executar, sendo muito parecido com o comportamento imperativo das linguagens naturais que expressam **ordens**.
- O nome do paradigma, **imperativo**, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo...
- Exemplos: paradigmas
 - **procedimental**: C, Pascal, etc, e
 - **orientado a objetos**: Smalltalk, Java, C++, Python, etc

Programação declarativa

- Descreve **o que** o programa faz e não *como* seus procedimentos funcionam.
- Ênfase nos **resultados**, no que se deseja obter.
- Exemplos: paradigmas
 - **funcional**: Haskell, OCaml, LISP, etc, e
 - **lógico**: Prolog, etc.

- 1 Paradigmas de programação
- 2 Programação funcional**
- 3 A crise do software
- 4 Algumas características de Haskell
- 5 Antecedentes históricos
- 6 Algumas empresas que usam Haskell
- 7 Curso online
- 8 Programando com Funções
- 9 Experimentando Haskell

Programação funcional

- **Programação funcional** é um paradigma de programação que descreve uma computação como uma **expressão** a ser **avaliada**.
- A principal forma de estruturar o programa é pela definição e aplicação de **funções**.

Exemplo: quick sort em C

```
// To sort array a[] of size n: qsort(a, 0, n-1)
```

```
void qsort(int a[], int lo, int hi) {  
    int h, l, p, t;  
  
    if (lo < hi) {  
        l = lo;  
        h = hi;  
        p = a[hi];  
  
        do {  
            while ((l < h) && (a[l] <= p))  
                l = l+1;  
            while ((h > l) && (a[h] >= p))  
                h = h-1;  
            if (l < h) {  
                t = a[l];  
                a[l] = a[h];  
                a[h] = t;  
            }  
        } while (l < h);  
  
        a[hi] = a[l];  
        a[l] = p;  
  
        qsort(a, lo, l-1);  
        qsort(a, l+1, hi);  
    }  
}
```

Exemplo: quick sort em Haskell

```
qsort [] = []  
qsort (primeiro:resto) = qsort (filter (<primeiro) resto) ++  
                          [primeiro] ++  
                          qsort (filter (>primeiro) resto)
```

Observações:

- o símbolo `=` separa os lados esquerdo e direito de uma equação, usada para definir variáveis e funções.
- `[]` denota a lista vazia.
- `primeiro:resto` denota uma lista não vazia cuja cabeça é `primeiro` e cuja cauda é `resto`.
- A sintaxe para aplicação de função consiste em escrever a função seguida dos argumentos, separados por espaços, como em `max 10 (2+x)`.
- A função `filter` seleciona os elementos de uma lista que satisfazem uma determinada propriedade (expressa por uma função que retorna *verdadeiro* ou *falso*).
- `(<primeiro)` e `(>primeiro)` são funções que verificam se o seu argumento é menor ou maior do que `primeiro`, respectivamente. São funções anônimas construídas pela aplicação parcial dos operadores `<` e `>`.
- O operador `++` concatena duas listas.
- Uma lista pode ser escrita enumerando os seus elementos separados por vírgula e colocados entre colchetes.

- 1 Paradigmas de programação
- 2 Programação funcional
- 3 A crise do software
- 4 Algumas características de Haskell
- 5 Antecedentes históricos
- 6 Algumas empresas que usam Haskell
- 7 Curso online
- 8 Programando com Funções
- 9 Experimentando Haskell

A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma **clara, concisa**, e com um **alto nível de abstração**;
 - suportam componentes de software **reutilizáveis**;
 - incentivam o uso de **verificação formal**;
 - permitem **prototipagem rápida**;
 - fornecem **poderosas ferramentas** de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o **tamanho** e a **complexidade** dos programas de computador modernos
 - o **tempo** e o **custo** de desenvolvimento dos programas
 - a **confiança** de que os programas já concluídos funcionam **corretamente**



As **linguagens funcionais** oferecem um quadro particularmente *elegante* para abordar estes objetivos.

A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma **clara, concisa**, e com um **alto nível de abstração**;
 - suportam componentes de software **reutilizáveis**;
 - incentivam o uso de **verificação formal**;
 - permitem **prototipagem rápida**;
 - fornecem **poderosas ferramentas** de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o **tamanho** e a **complexidade** dos programas de computador modernos
 - o **tempo** e o **custo** de desenvolvimento dos programas
 - a **confiança** de que os programas já concluídos funcionam **corretamente**



As **linguagens funcionais** oferecem um quadro particularmente *elegante* para abordar estes objetivos.

A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma **clara, concisa**, e com um **alto nível de abstração**;
 - suportam componentes de software **reutilizáveis**;
 - incentivam o uso de **verificação formal**;
 - permitem **prototipagem rápida**;
 - fornecem **poderosas ferramentas** de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o **tamanho** e a **complexidade** dos programas de computador modernos
 - o **tempo** e o **custo** de desenvolvimento do programas
 - a **confiança** de que os programas já concluídos funcionam **corretamente**



As **linguagens funcionais** oferecem um quadro particularmente *elegante* para abordar estes objetivos.

A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma **clara, concisa**, e com um **alto nível de abstração**;
 - suportam componentes de software **reutilizáveis**;
 - incentivam o uso de **verificação formal**;
 - permitem **prototipagem rápida**;
 - fornecem **poderosas ferramentas** de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o **tamanho** e a **complexidade** dos programas de computador modernos
 - o **tempo** e o **custo** de desenvolvimento dos programas
 - a **confiança** de que os programas já concluídos funcionam **corretamente**



As **linguagens funcionais** oferecem um quadro particularmente *elegante* para abordar estes objetivos.

A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma **clara, concisa**, e com um **alto nível de abstração**;
 - suportam componentes de software **reutilizáveis**;
 - incentivam o uso de **verificação formal**;
 - permitem **prototipagem rápida**;
 - fornecem **poderosas ferramentas** de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o **tamanho** e a **complexidade** dos programas de computador modernos
 - o **tempo** e o **custo** de desenvolvimento dos programas
 - a **confiança** de que os programas já concluídos funcionam **corretamente**



As **linguagens funcionais** oferecem um quadro particularmente *elegante* para abordar estes objetivos.

A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma **clara, concisa**, e com um **alto nível de abstração**;
 - suportam componentes de software **reutilizáveis**;
 - incentivam o uso de **verificação formal**;
 - permitem **prototipagem rápida**;
 - fornecem **poderosas ferramentas** de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o **tamanho** e a **complexidade** dos programas de computador modernos
 - o **tempo** e o **custo** de desenvolvimento dos programas
 - a **confiança** de que os programas já concluídos funcionam **corretamente**



As **linguagens funcionais** oferecem um quadro particularmente *elegante* para abordar estes objetivos.

A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma **clara, concisa**, e com um **alto nível de abstração**;
 - suportam componentes de software **reutilizáveis**;
 - incentivam o uso de **verificação formal**;
 - permitem **prototipagem rápida**;
 - fornecem **poderosas ferramentas** de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o **tamanho** e a **complexidade** dos programas de computador modernos
 - o **tempo** e o **custo** de desenvolvimento do programas
 - a **confiança** de que os programas já concluídos funcionam **corretamente**



As **linguagens funcionais** oferecem um quadro particularmente *elegante* para abordar estes objetivos.

A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma **clara, concisa**, e com um **alto nível de abstração**;
 - suportam componentes de software **reutilizáveis**;
 - incentivam o uso de **verificação formal**;
 - permitem **prototipagem rápida**;
 - fornecem **poderosas ferramentas** de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o **tamanho** e a **complexidade** dos programas de computador modernos
 - o **tempo** e o **custo** de desenvolvimento do programas
 - a **confiança** de que os programas já concluídos funcionam **corretamente**



As **linguagens funcionais** oferecem um quadro particularmente *elegante* para abordar estes objetivos.

A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma **clara**, **concisa**, e com um **alto nível de abstração**;
 - suportam componentes de software **reutilizáveis**;
 - incentivam o uso de **verificação formal**;
 - permitem **prototipagem rápida**;
 - fornecem **poderosas ferramentas** de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o **tamanho** e a **complexidade** dos programas de computador modernos
 - o **tempo** e o **custo** de desenvolvimento do programas
 - a **confiança** de que os programas já concluídos funcionam **corretamente**



As **linguagens funcionais** oferecem um quadro particularmente *elegante* para abordar estes objetivos.

A Crise do Software

- *Linguagens declarativas* (incluindo linguagens funcionais):
 - permitem que programas sejam escritos de forma **clara, concisa**, e com um **alto nível de abstração**;
 - suportam componentes de software **reutilizáveis**;
 - incentivam o uso de **verificação formal**;
 - permitem **prototipagem rápida**;
 - fornecem **poderosas ferramentas** de resolução de problemas.
- Estas características podem ser úteis na abordagem de dificuldades encontradas no desenvolvimento de software:
 - o **tamanho** e a **complexidade** dos programas de computador modernos
 - o **tempo** e o **custo** de desenvolvimento do programas
 - a **confiança** de que os programas já concluídos funcionam **corretamente**



As **linguagens funcionais** oferecem um quadro particularmente *elegante* para abordar estes objetivos.

- 1 Paradigmas de programação
- 2 Programação funcional
- 3 A crise do software
- 4 Algumas características de Haskell
- 5 Antecedentes históricos
- 6 Algumas empresas que usam Haskell
- 7 Curso online
- 8 Programando com Funções
- 9 Experimentando Haskell

Algumas características de Haskell

- Programas são concisos
- Tipagem estática
- Sistema de tipos poderoso
- Polimorfismo paramétrico
- Tipos recursivos e funções recursivas
- Funções de ordem superior
- Linguagem pura (declarativa)
- Avaliação *lazy*
- Maior facilidade de prova de propriedades de programas

- 1 Paradigmas de programação
- 2 Programação funcional
- 3 A crise do software
- 4 Algumas características de Haskell
- 5 **Antecedentes históricos**
- 6 Algumas empresas que usam Haskell
- 7 Curso online
- 8 Programando com Funções
- 9 Experimentando Haskell

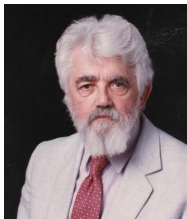
- Década de 1930:



Alonzo Church desenvolve o **cálculo lambda**, uma teoria de funções simples, mas poderosa.

Antecedentes históricos (cont.)

- Década de 1950:



John McCarthy desenvolve **Lisp**, a primeira linguagem funcional, com algumas influências do **cálculo lambda**, mas mantendo as atribuições de variáveis.

Antecedentes históricos (cont.)

- Década de 1960:



Peter Landin desenvolve **ISWIM**, a primeira linguagem funcional pura, baseada fortemente no **cálculo lambda**, sem atribuições.

- Década de 1970:



John Backus desenvolve **FP**, uma linguagem funcional que enfatiza funções de ordem superior e raciocínio sobre programas.

Antecedentes históricos (cont.)

- Década de 1970:



Robin Milner e outros desenvolvem **ML**, a primeira linguagem funcional moderna, que introduziu a inferência de tipos e tipos polimórficos.

Antecedentes históricos (cont.)

- Décadas de 1970 e 1980:



David Turner desenvolve uma série de linguagens funcionais com *avaliação lazy*, culminando com o sistema **Miranda**. Porém Miranda não é de domínio público.

- 1987:

Haskell

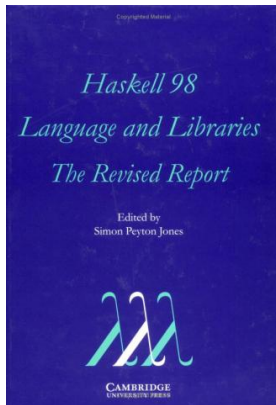
A Purely Functional Language

Um comitê internacional de pesquisadores inicia o desenvolvimento de **Haskell**, com objetivo de se obter uma linguagem funcional *lazy* padrão aberta. Isso consolidaria as linguagens existentes, servindo como base para pesquisas futuras no desenvolvimento de linguagens funcionais.

- 1990:
Definição da primeira versão de Haskell.

Antecedentes históricos (cont.)

- 2003:



O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem Haskell.

- 2009:



O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem Haskell.

- 1 Paradigmas de programação
- 2 Programação funcional
- 3 A crise do software
- 4 Algumas características de Haskell
- 5 Antecedentes históricos
- 6 Algumas empresas que usam Haskell**
- 7 Curso online
- 8 Programando com Funções
- 9 Experimentando Haskell

Quem usa Haskell?

- Exemplos de empresas que usam Haskell:
 - **ABN AMRO** análise de riscos financeiros
 - **AT&T** automatização de processamento de formulários
 - **Bank of America Merrill Lynch** transformação de dados
 - **Bump** servidores baseados em Haskell
 - **Facebook** manipulação da base de código PHP
 - **Google** infra-estrutura interna de TI
 - **MITRE** análise de protocolos de criptografia
 - **NVIDIA** ferramentas usadas internamente
 - **Qualcomm, Inc** geração de interfaces de programação para Lua
 - **The New York Times** processamento de imagens
- Para maiores detalhes visite a página *Haskell na indústria* em http://www.haskell.org/haskellwiki/Haskell_in_industry.
- Uma lista de empresas que usam Haskell: <https://github.com/erkmos/haskell-companies>

- 1 Paradigmas de programação
- 2 Programação funcional
- 3 A crise do software
- 4 Algumas características de Haskell
- 5 Antecedentes históricos
- 6 Algumas empresas que usam Haskell
- 7 Curso online**
- 8 Programando com Funções
- 9 Experimentando Haskell



- **Functional Programming in Haskell: Supercharge Your Coding**

- Universidade de Glasgow

- Início: 2 de abril de 2018

- Duração: 6 semanas

- Dedicação: 4 horas por semana

- **https:**

[//www.futurelearn.com/courses/functional-programming-haskell](https://www.futurelearn.com/courses/functional-programming-haskell)

- 1 Paradigmas de programação
- 2 Programação funcional
- 3 A crise do software
- 4 Algumas características de Haskell
- 5 Antecedentes históricos
- 6 Algumas empresas que usam Haskell
- 7 Curso online
- 8 Programando com Funções**
- 9 Experimentando Haskell

Função

- **Função** é uma **relação** que associa cada valor de um domínio (argumento) a um único valor de um contra-domínio (resultado).
- Em geral o mapeamento é estabelecido por uma **lei de associação**, expressa por um **algoritmo**.
- Em Haskell uma função é **definida** usando uma ou mais **equações**.
- Uma **equação** especifica:
 - o **nome** da função,
 - os **parâmetros formais**, que representam os **argumentos**, e
 - o **corpo**, que especifica o resultado em termos dos parâmetros formais.
- **Exemplo:**
Uma função chamada **dobro** que recebe um número x como argumento, e produz o resultado $x + x$:

```
dobro x = x + x
```

Função

- **Função** é uma **relação** que associa cada valor de um domínio (argumento) a um único valor de um contra-domínio (resultado).
- Em geral o mapeamento é estabelecido por uma **lei de associação**, expressa por um **algoritmo**.
- Em Haskell uma função é **definida** usando uma ou mais **equações**.
- Uma **equação** especifica:
 - o **nome** da função,
 - os **parâmetros formais**, que representam os **argumentos**, e
 - o **corpo**, que especifica o resultado em termos dos parâmetros formais.
- **Exemplo:**

Uma função chamada **dobro** que recebe um número x como argumento, e produz o resultado $x + x$:

```
dobro x = x + x
```

Função

- **Função** é uma **relação** que associa cada valor de um domínio (argumento) a um único valor de um contra-domínio (resultado).
- Em geral o mapeamento é estabelecido por uma **lei de associação**, expressa por um **algoritmo**.
- Em Haskell uma função é **definida** usando uma ou mais **equações**.
- Uma **equação** especifica:
 - o **nome** da função,
 - os **parâmetros formais**, que representam os **argumentos**, e
 - o **corpo**, que especifica o resultado em termos dos parâmetros formais.
- **Exemplo:**

Uma função chamada **dobro** que recebe um número x como argumento, e produz o resultado $x + x$:

```
dobro x = x + x
```

Função

- **Função** é uma **relação** que associa cada valor de um domínio (argumento) a um único valor de um contra-domínio (resultado).
- Em geral o mapeamento é estabelecido por uma **lei de associação**, expressa por um **algoritmo**.
- Em Haskell uma função é **definida** usando uma ou mais **equações**.
- Uma **equação** especifica:
 - o **nome** da função,
 - os **parâmetros formais**, que representam os **argumentos**, e
 - o **corpo**, que especifica o resultado em termos dos parâmetros formais.
- **Exemplo:**
Uma função chamada **dobro** que recebe um número x como argumento, e produz o resultado $x + x$:

```
dobro x = x + x
```

Função

- **Função** é uma **relação** que associa cada valor de um domínio (argumento) a um único valor de um contra-domínio (resultado).
- Em geral o mapeamento é estabelecido por uma **lei de associação**, expressa por um **algoritmo**.
- Em Haskell uma função é **definida** usando uma ou mais **equações**.
- Uma **equação** especifica:
 - o **nome** da função,
 - os **parâmetros formais**, que representam os **argumentos**, e
 - o **corpo**, que especifica o resultado em termos dos parâmetros formais.
- Exemplo:

Uma função chamada **dobro** que recebe um número x como argumento, e produz o resultado $x + x$:

```
dobro x = x + x
```

Função

- **Função** é uma **relação** que associa cada valor de um domínio (argumento) a um único valor de um contra-domínio (resultado).
- Em geral o mapeamento é estabelecido por uma **lei de associação**, expressa por um **algoritmo**.
- Em Haskell uma função é **definida** usando uma ou mais **equações**.
- Uma **equação** especifica:
 - o **nome** da função,
 - os **parâmetros formais**, que representam os **argumentos**, e
 - o **corpo**, que especifica o resultado em termos dos parâmetros formais.
- Exemplo:

Uma função chamada **dobro** que recebe um número x como argumento, e produz o resultado $x + x$:

```
dobro x = x + x
```

Função

- **Função** é uma **relação** que associa cada valor de um domínio (argumento) a um único valor de um contra-domínio (resultado).
- Em geral o mapeamento é estabelecido por uma **lei de associação**, expressa por um **algoritmo**.
- Em Haskell uma função é **definida** usando uma ou mais **equações**.
- Uma **equação** especifica:
 - o **nome** da função,
 - os **parâmetros formais**, que representam os **argumentos**, e
 - o **corpo**, que especifica o resultado em termos dos parâmetros formais.

- **Exemplo:**

Uma função chamada **dobro** que recebe um número x como argumento, e produz o resultado $x + x$:

```
dobro x = x + x
```

Função

- **Função** é uma **relação** que associa cada valor de um domínio (argumento) a um único valor de um contra-domínio (resultado).
- Em geral o mapeamento é estabelecido por uma **lei de associação**, expressa por um **algoritmo**.
- Em Haskell uma função é **definida** usando uma ou mais **equações**.
- Uma **equação** especifica:
 - o **nome** da função,
 - os **parâmetros formais**, que representam os **argumentos**, e
 - o **corpo**, que especifica o resultado em termos dos parâmetros formais.
- **Exemplo:**
Uma função chamada **dobro** que recebe um número x como argumento, e produz o resultado $x + x$:

```
dobro x = x + x
```

Aplicação de função

- Quando uma função é **aplicada** aos argumentos atuais, o resultado é obtido pela **substituição** desses argumentos no corpo da função no lugar dos nomes dos argumentos.
- Este processo pode produzir imediatamente um resultado que não pode ser mais simplificado, como por exemplo um número.
- Mais comumente, no entanto, a substituição produz uma expressão contendo outras aplicações de função, que devem, então, ser processadas da mesma maneira para produzir o resultado final.
- Em geral, a **ordem na qual funções são aplicadas** em um cálculo
 - não afeta o **valor do resultado final**, mas
 - pode afetar o **número de passos necessários**, e
 - pode determinar se o processo de cálculo **termina** ou **não termina**.

Aplicação de função

- Quando uma função é **aplicada** aos argumentos atuais, o resultado é obtido pela **substituição** desses argumentos no corpo da função no lugar dos nomes dos argumentos.
- Este processo pode produzir imediatamente um resultado que não pode ser mais simplificado, como por exemplo um número.
- Mais comumente, no entanto, a substituição produz uma expressão contendo outras aplicações de função, que devem, então, ser processadas da mesma maneira para produzir o resultado final.
- Em geral, a **ordem na qual funções são aplicadas** em um cálculo
 - não afeta o valor do resultado final, mas
 - pode afetar o número de passos necessários, e
 - pode determinar se o processo de cálculo **termina** ou **não termina**.

Aplicação de função

- Quando uma função é **aplicada** aos argumentos atuais, o resultado é obtido pela **substituição** desses argumentos no corpo da função no lugar dos nomes dos argumentos.
- Este processo pode produzir imediatamente um resultado que não pode ser mais simplificado, como por exemplo um número.
- Mais comumente, no entanto, a substituição produz uma expressão contendo outras aplicações de função, que devem, então, ser processadas da mesma maneira para produzir o resultado final.
- Em geral, a **ordem na qual funções são aplicadas** em um cálculo
 - não afeta o valor do resultado final, mas
 - pode afetar o número de passos necessários, e
 - pode determinar se o processo de cálculo **termina** ou **não termina**.

Aplicação de função

- Quando uma função é **aplicada** aos argumentos atuais, o resultado é obtido pela **substituição** desses argumentos no corpo da função no lugar dos nomes dos argumentos.
- Este processo pode produzir imediatamente um resultado que não pode ser mais simplificado, como por exemplo um número.
- Mais comumente, no entanto, a substituição produz uma expressão contendo outras aplicações de função, que devem, então, ser processadas da mesma maneira para produzir o resultado final.
- Em geral, a **ordem na qual funções são aplicadas** em um cálculo
 - não afeta o **valor do resultado final**, mas
 - pode afetar o **número de passos** necessários, e
 - pode determinar se o processo de cálculo **termina** ou **não termina**.

Aplicação de função

- Quando uma função é **aplicada** aos argumentos atuais, o resultado é obtido pela **substituição** desses argumentos no corpo da função no lugar dos nomes dos argumentos.
- Este processo pode produzir imediatamente um resultado que não pode ser mais simplificado, como por exemplo um número.
- Mais comumente, no entanto, a substituição produz uma expressão contendo outras aplicações de função, que devem, então, ser processadas da mesma maneira para produzir o resultado final.
- Em geral, a **ordem na qual funções são aplicadas** em um cálculo
 - não afeta o **valor do resultado final**, mas
 - pode afetar o **número de passos** necessários, e
 - pode determinar se o processo de cálculo **termina** ou **não termina**.

Aplicação de função

- Quando uma função é **aplicada** aos argumentos atuais, o resultado é obtido pela **substituição** desses argumentos no corpo da função no lugar dos nomes dos argumentos.
- Este processo pode produzir imediatamente um resultado que não pode ser mais simplificado, como por exemplo um número.
- Mais comumente, no entanto, a substituição produz uma expressão contendo outras aplicações de função, que devem, então, ser processadas da mesma maneira para produzir o resultado final.
- Em geral, a **ordem na qual funções são aplicadas** em um cálculo
 - não afeta o **valor do resultado final**, mas
 - pode afetar o **número de passos** necessários, e
 - pode determinar se o processo de cálculo **termina** ou **não termina**.

Aplicação de função

- Quando uma função é **aplicada** aos argumentos atuais, o resultado é obtido pela **substituição** desses argumentos no corpo da função no lugar dos nomes dos argumentos.
- Este processo pode produzir imediatamente um resultado que não pode ser mais simplificado, como por exemplo um número.
- Mais comumente, no entanto, a substituição produz uma expressão contendo outras aplicações de função, que devem, então, ser processadas da mesma maneira para produzir o resultado final.
- Em geral, a **ordem na qual funções são aplicadas** em um cálculo
 - não afeta o **valor do resultado final**, mas
 - pode afetar o **número de passos** necessários, e
 - pode determinar se o processo de cálculo **termina** ou **não termina**.

Exemplos: aplicação de função

- **Exemplo:**

Aplicação da função `dobro` no argumento 3

```
dobro 3  
= { aplicando dobro }  
3 + 3  
= { aplicando + }  
6
```

- Exemplo:

Aplicação aninhada de `dobro`

```
dobro (dobro 2)
= { aplicando dobro interno }
dobro (2 + 2)
= { aplicando + }
dobro 4
= { aplicando dobro }
4 + 4
= { aplicando + }
8
```

Exemplos: aplicação de função (cont.)

- Exemplo:

Aplicação aninhada de `dobro`, calculada de outra maneira

```
dobro (dobro 2)
= { aplicando dobro externo }
dobro 2 + dobro 2
= { aplicando o primeiro dobro }
(2 + 2) + dobro 2
= { aplicando o primeiro + }
4 + dobro 2
= { aplicando dobro }
4 + (2 + 2)
= { aplicando o segundo + }
4 + 4
= { aplicando + }
8
```

O que é uma linguagem funcional?

As opiniões divergem, e é difícil dar uma definição precisa, mas de um modo geral:

- **Programação funcional** é um estilo de programação em que o método básico de computação é a **aplicação de funções** a argumentos.
- Uma **linguagem funcional** é aquela que apoia e incentiva o estilo funcional.

O que é uma linguagem funcional? (cont.)

Exemplo:

Somando os inteiros 1 a 10 em C:

```
int total = 0;
for (int i = 1; i <= 10; ++i)
    total = total + i;
```

- O método de cálculo é **atribuição de variável**.
- Em geral, linguagens de programação em que o método básico de computação consiste em mudar os valores armazenados em variáveis são chamadas de **linguagens imperativas**, pois os programas nestas linguagens são construídos a partir de instruções imperativas que especificam precisamente **como o cálculo deve ser realizado**.

O que é uma linguagem funcional? (cont.)

Exemplo:

Somando os inteiros 1 a 10 em Haskell:

```
sum [1..10]
```

- O método de cálculo é **aplicação de função**.

- 1 Paradigmas de programação
- 2 Programação funcional
- 3 A crise do software
- 4 Algumas características de Haskell
- 5 Antecedentes históricos
- 6 Algumas empresas que usam Haskell
- 7 Curso online
- 8 Programando com Funções
- 9 Experimentando Haskell

Experimentando Haskell

```
sum :: Num a => [a] -> a
```

```
sum [] = 0
```

```
sum (x:xs) = x + sum xs
```

```
sum [1, 2, 3]
```

```
= { aplicando sum }
```

```
1 + sum [2, 3]
```

```
= { aplicando sum }
```

```
1 + (2 + sum [3])
```

```
= { aplicando sum }
```

```
1 + (2 + (3 + sum []))
```

```
= { aplicando sum }
```

```
1 + (2 + (3 + 0))
```

```
= { aplicando + }
```

```
6
```

```
qsort []      = []  
qsort (x:xs) = qsort ys ++ [x] ++ qsort zs  
    where  
        ys = [a | a <- xs, a <= x]  
        zs = [b | b <- xs, b > x]
```

?

```
qsort [x]
= { aplicando qsort }
qsort [] ++ [x] ++ qsort []
= { aplicando qsort }
[] ++ [x] ++ []
= { aplicando ++ }
[x]
```

Experimentando Haskell (cont.)

```
qsort [3, 5, 1, 4, 2]
= { aplicando qsort }
qsort [1, 2] ++ [3] ++ qsort [5, 4]
= { aplicando qsort }
(qsort [] ++ [1] ++ qsort [2]) ++ [3] ++ (qsort [4] ++ [5] ++ qsort [])
= { aplicando qsort, propriedade anterior }
([] ++ [1] ++ [2]) ++ [3] ++ ([4] ++ [5] ++ [])
= { aplicando ++ }
[1, 2] ++ [3] ++ [4, 5]
= { aplicando ++ }
[1, 2, 3, 4, 5]
```

- O tipo de `qsort` é

```
qsort :: Ord a => [a] -> [a]
```

- Dada uma lista `xs`, `qsort xs` é a lista formada pelos elementos de `xs` em ordem crescente.
- `qsort` implementa o algoritmo de ordenação **quick sort**.

Exercícios

Exercise 1

Dê outro cálculo possível para o resultado de `dobro (dobro 2)`.

Exercise 2

Mostre que `sum [x] = x` para qualquer número `x`.

Exercise 3

Defina uma função `product` que produza o produto de uma lista de números, e mostre, usando sua definição, que `product [2,3,4] = 24`

Exercise 4

Mostre como a definição da função `qsort` deve ser modificada para que ela produza uma versão de uma lista ordenada em ordem decrescente.

Exercise 5

Qual é o efeito de trocar `<=` por `<` na definição de `qsort`?

Dica:

considere o exemplo `qsort [2,2,3,1,1]`.

Fim