

Modelos de Linguagem de Programação

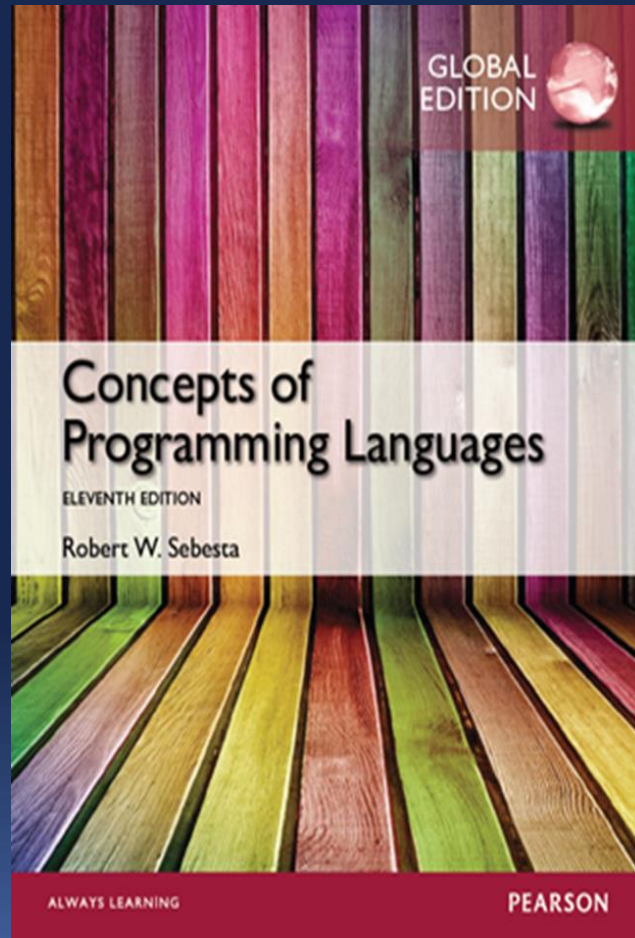
Unidade 6 – Paradigma Funcional de Programação



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

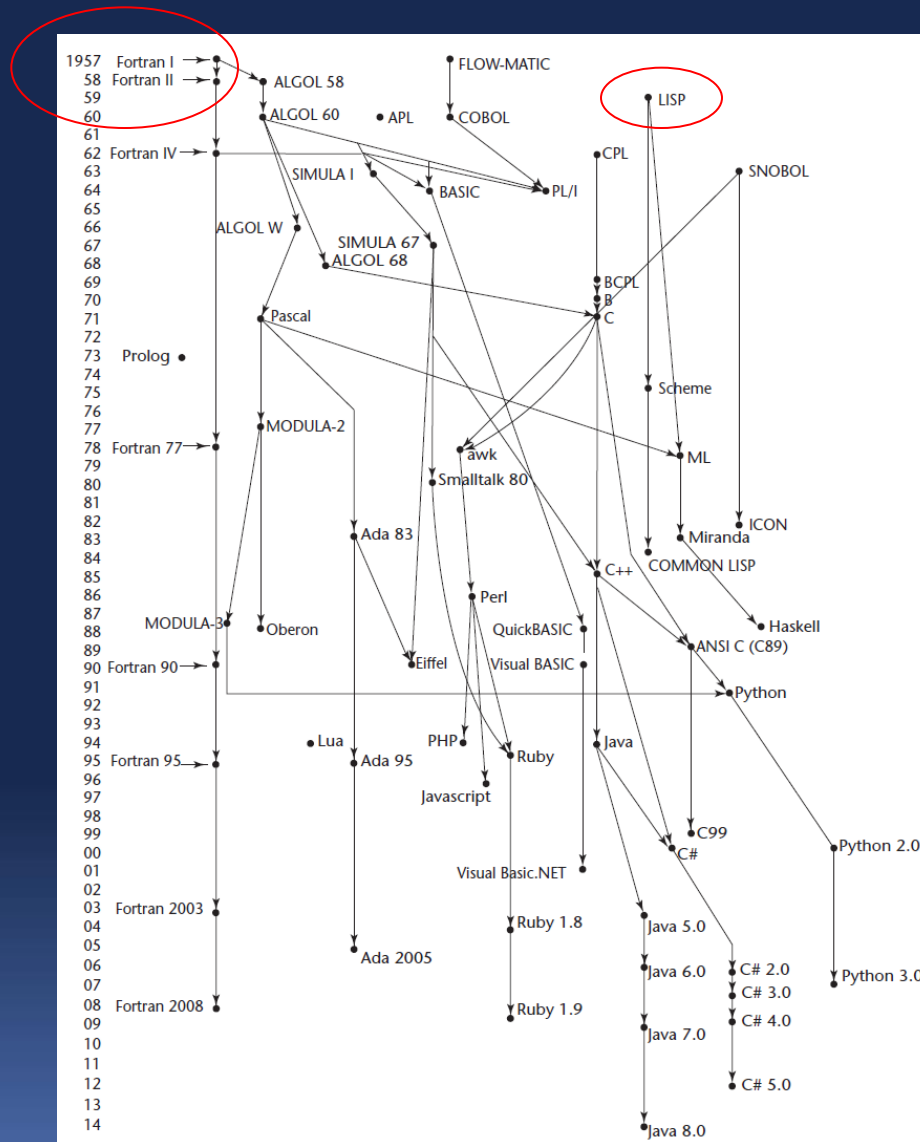
Bibliografia

- R. Sebesta – Concepts Of Programming Languages



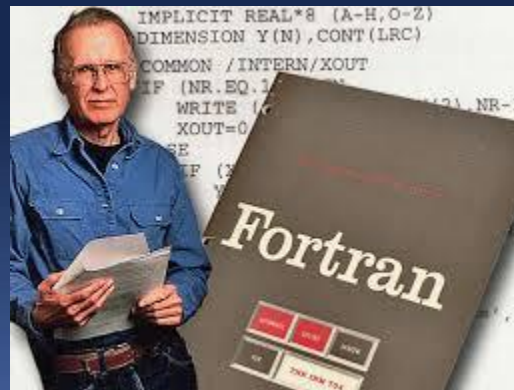
O Paradigma Funcional

- ✓ A maioria das linguagens de Programação tiveram suas origens no Fortran.



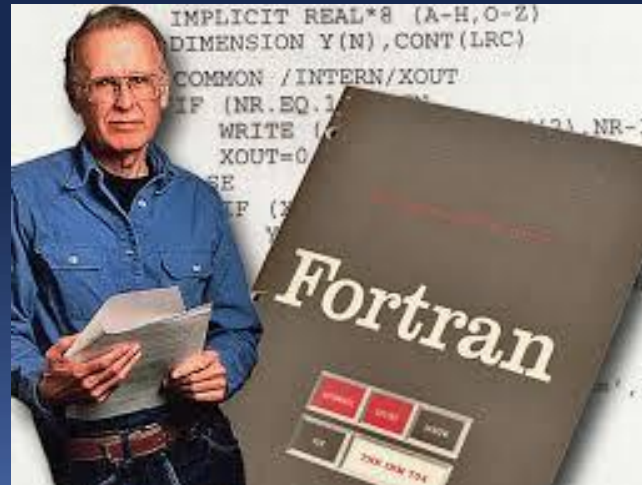
O Paradigma Funcional

- ✓ **Fortran** foi a primeira linguagem de programação;
- ✓ O primeiro compilador Fortran foi desenvolvido por uma equipe da IBM sendo chefiada por **John Backus**, na década de 50;
- ✓ Linguagem fortemente **aderente** ao paradigma **imperativo** e seu projeto tinha como grande objetivo o uso eficiente de máquina;
- ✓ A partir da Linguagem **Fortran**, diversas outras linguagens foram desenvolvidas;
- ✓ Mas, em **1977** na palestra ministrada por **John Backus** quando ganhou o prêmio **ACM Turing**, ele argumentou que **linguagens funcionais** são melhores que as **imperativas**, pois podem apresentar mais **confiabilidade**, **legibilidade** e com maior probabilidade de estarem corretas.





Em que se baseou Backus para afirmar que Linguagens Funcionais apresentam maior legibilidade e confiabilidade ?





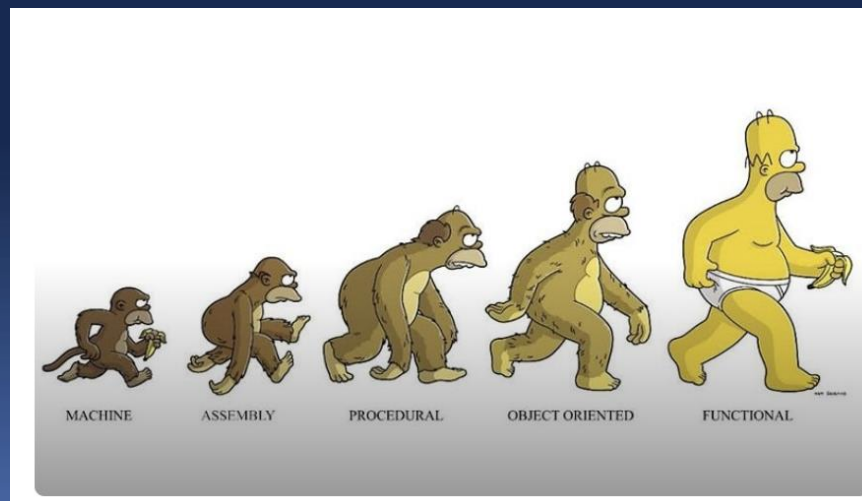
Linguagens Funcionais

- ✓ O **argumento** de **John Backus** teve como base que em **Linguagens Puramente Funcionais**, o significado das expressões são **independentes** de **contexto**;
- ✓ Em **Linguagens Puramente Funcionais** nem expressões nem funções apresentam Efeitos Colaterais (**Side Effects**);
- ✓ **Backus** propôs na época uma nova Linguagem Funcional chamada **FP** (Functional Programming) para embasar seu argumento;
- ✓ A linguagem **não vingou**, mas abriu espaço para **pesquisa** do **Paradigma Funcional** de Programação;



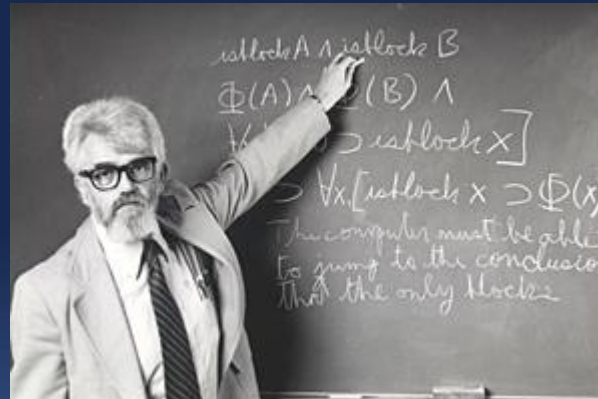
Programas Funcionais x Imperativos

- ✓ Uma das principais características dos programas escritos em Linguagens Imperativas é que eles possuem **ESTADO**;
- ✓ Ou seja, a **execução** do programa corresponde à sucessivas **mudanças de estado** em **variáveis**, no qual a resposta do programa será representada pelos estados finais de suas **variáveis** (Transformação de Estado);
- ✓ Para grandes programas, esta tarefa pode ser difícil;
- ✓ Em programas escritos com **Linguagens Puramente Funcionais** estes problemas não ocorrem, pois programas funcionais **NÃO** possuem **Estado** nem tão pouco **Variáveis**;



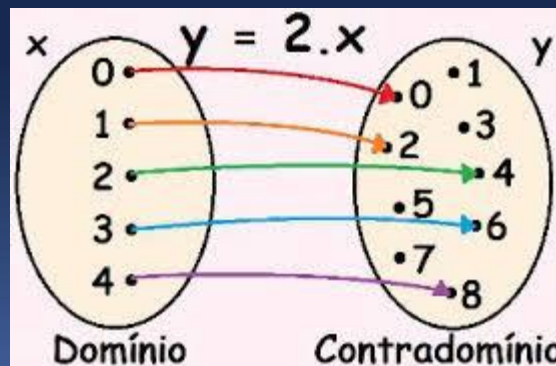
Linguagem de Programação Lisp

- ✓ Concebida por **John McCarthy** em **1959**, no **MIT**;
- ✓ Focada no uso exclusivo de **funções matemáticas** como estrutura de dados;
- ✓ Tem como base formal o **Cálculo Lambda** de Alonzo Church;
- ✓ É **a mais importante linguagem** representativa do **Paradigma Funcional**.



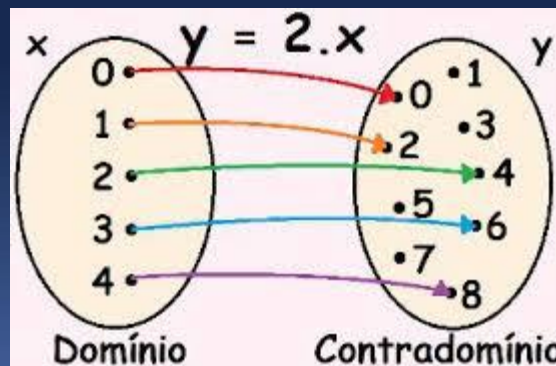
Funções Matemáticas

- ✓ Uma **função** matemática é um **mapeamento** de elementos de um conjunto, chamado conjunto **Domínio**, para outro conjunto, chamado **Contra-Domínio** (range set);
- ✓ A definição de uma função especifica o domínio e o range set, de forma **explícita** ou **implícita**.
- ✓ O **mapeamento** é descrito por uma **expressão**;
- ✓ **Funções** são geralmente aplicadas a um elemento específico do Domínio, passado como parâmetro para a função;
- ✓ Ao se aplicar o parâmetro à função obtém-se um valor do Contra-Domínio.



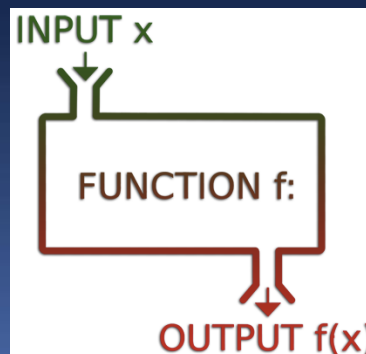
Funções Matemáticas

- ✓ Uma **função** matemática é um **mapeamento** de elementos de um conjunto, chamado conjunto **Domínio**, para outro conjunto, chamado **Contra-Domínio** (range set);
- ✓ A definição de uma função especifica o domínio e o range set, de forma **explícita** ou **implícita**.
- ✓ O **mapeamento** é descrito por uma **expressão**;
- ✓ **Funções** são geralmente aplicadas a um elemento específico do Domínio, passado como parâmetro para a função;
- ✓ Ao se aplicar o parâmetro à função obtém-se um valor do Contra-Domínio.



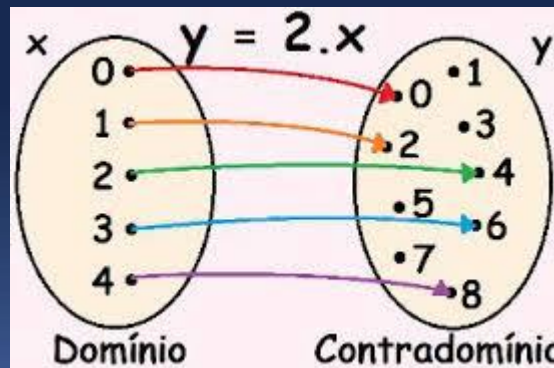
Funções Matemáticas

- ✓ A **ordem** de avaliação da expressão de mapeamento é **controlada** por **Recursão** e **expressões condicionais**;
- ✓ Diferentemente do **Paradigma Imperativo** no qual a avaliação é feita por **sequenciamento** e **repetição iterativa**;
- ✓ Outra importante característica das **funções matemáticas** é que elas **sempre mapeiam o mesmo valor do Contra-Domínio para um valor do Domínio**;
- ✓ Essa característica ocorre pois com funções matemáticas **não** há **dependência de valores externos** e **não há side effects**;
- ✓ Essa particularidade das funções matemáticas **não** ocorre nas linguagens imperativas, pois nestas linguagens um **subprograma** pode depender dos valores **correntes** de diversas **variáveis não-locais** ou **globais**, causando assim **side effects**.



Funções Matemáticas

- ✓ Na **Matemática**, **não** existe algo como uma **variável** que é usada para modelar uma localização de memória;
- ✓ **Variáveis locais** nas **linguagens imperativas** mantêm o **estado** da função. Nessas linguagens, a computação é realizada pela avaliação das expressões em comandos de atribuição que modificam o estado do programa;
- ✓ Na **Matemática**, por outro lado, **não** existe o conceito de **estado de uma função**;
- ✓ Uma **função matemática sempre** mapeia seu parâmetro (ou parâmetros) para um valor (ou valores) ao invés de especificar uma sequência de operações em valores de memória para computar um resultado.





Funções Simples

- ✓ Definições de funções são frequentemente escritas por: nome da função, seguido por uma lista de parâmetros entre parênteses, seguidas por uma **expressão** de mapeamento;
- ✓ Por exemplo:

$\text{cubo}(x) \equiv x * x * x$, onde x é um número real.

- ✓ Nessa função, o Domínio e o Contra-Domínio são o conjunto dos reais;
- ✓ O símbolo \equiv é usado para “**é definida como**”;
- ✓ O parâmetro x representa qualquer elemento \in ao domínio, mas é **fixado** para representar um **valor específico** durante a **avaliação** da expressão;
- ✓ Esta é a forma pela qual os **parâmetros** das funções matemáticas **diferem** das **variáveis** nas linguagens **imperativas**.





Aplicações de Funções

- ✓ **Aplicações de funções** são especificadas pelo **emparelhamento** do nome da função com um elemento fixo e particular do Domínio;
- ✓ O elemento do Contra-Domínio é obtido pela **avaliação** da **expressão** de mapeamento com o valor do domínio substituído pelas ocorrências do parâmetro;
- ✓ É importante observar, mais uma vez, que durante a avaliação, **toda ocorrência de um parâmetro é ligada (bound)** a um valor do domínio, sendo portanto constante durante toda a avaliação;
- ✓ Exemplo:

$$\text{cubo}(x) \equiv x * x * x$$

Definição

$$\text{cubo}(2.0) \equiv 2.0 * 2.0 * 2.0 = 8$$

Aplicação



Aplicações de Funções

- ✓ O parâmetro **x** é ligado (**bound**) ao valor **2.0** durante a avaliação;
- ✓ Não há parâmetros **sem ligação** (**unbound**);
- ✓ Adicionalmente, **x** é uma **constante** (seu valor não pode ser alterado) durante a avaliação.
- ✓ Assim, o argumento passado à função é **imutável**!
- ✓ Esse conceito é chamado **Imutabilidade**.

$$\text{cubo}(x) \equiv x * x * x$$

$$\text{cubo}(2.0) \equiv 2.0 * 2.0 * 2.0 = 8$$

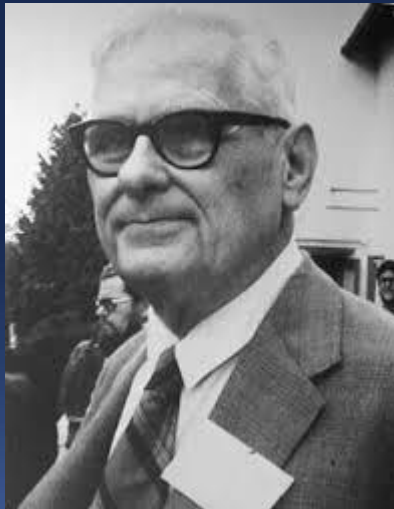
Cálculo Lambda

- ✓ A **base teórica** do Paradigma Funcional está no Cálculo Lambda;
- ✓ Nessa teoria, funções são definidas **sem nome** (**Notação Lambda**);
- ✓ Uma expressão lambda especifica os parâmetros e o mapeamento de uma função;
- ✓ A expressão lambda, portanto, é a própria função (sem nome);
- ✓ Por exemplo:

$$\lambda(x) x * x * x$$


Cálculo Lambda

- ✓ Church (1941) definiu um **modelo formal de computação** (um sistema formal para definição de funções, aplicação de funções e recursão) com o emprego de expressões lambda;
- ✓ O Cálculo Lambda pode ser tipado ou não tipado;
- ✓ As linguagens puramente funcionais baseiam-se no Cálculo Lambda não tipado;





Notação Lambda para Aplicação de Funções

- ✓ Como dito anteriormente, **antes** da avaliação, um **parâmetro** representa qualquer elemento do Domínio;
- ✓ Mas, durante a avaliação, o parâmetro é ligado (bound) a um valor específico que permanece constante durante a avaliação;
- ✓ Quando uma expressão lambda é avaliada para um determinado parâmetro, diz-se nesse caso, que a expressão é aplicada a este parâmetro;
- ✓ O mecanismo usado nesse caso é o mesmo que o ocorre na avaliação de qualquer função;
- ✓ Exemplo:

$$(\lambda(x) x * x * x) (2.0) = 8.0$$





Higher Order Function

- ✓ Uma **higher-order-function**, ou forma funcional (**functional form**), é uma função que pode receber como **parâmetro uma** ou **mais funções**;
- ✓ **Higher-order-function** podem também **produzir funções** como resultado;
- ✓ **Higher-order-functions** podem assim, ter funções como parâmetros e produzir também funções como resultado da avaliação;





Higher Order Function – Exemplo

$$h \equiv f \circ g$$

✓ Se: $f(x) \equiv x + 2$

$$g(x) \equiv 3 * x$$

✓ Então h é definida por: $h(x) \equiv f(g(x))$, ou $h(x) \equiv (3 * x) + 2$

✓ Estou passando para a função $f(x)$ uma outra função $g(x)$;

✓ O parâmetro para $f(x)$ é a função $g(x)$.





Higher Order Function – Exemplo

- ✓ Seja **h** uma função que toma como parâmetro uma **função** simples;
- ✓ Se **h** for aplicada a uma **lista de parâmetros**, **h** aplicará seus parâmetros funcionais à cada um dos valores da lista de parâmetros e coletará os resultados em uma lista;
- ✓ A função **h** é denotada por **α** (alfa).
- ✓ Exemplo: $h(x) \equiv x * x$
então: $\alpha(h, (2, 3, 4))$ resultará $(4, 9, 16)$
- ✓ Na programação funcional, a função **h** é chamada **função de mapeamento**.



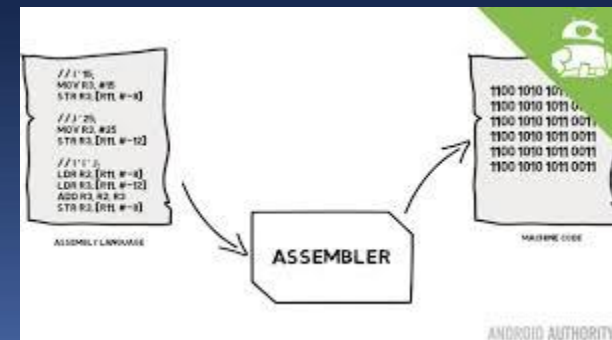
Objetivos da Programação Funcional

- ✓ **Focar** no emprego de **funções** matemáticas da forma mais **intensa** possível;
- ✓ Isso resulta numa abordagem totalmente diferente da Programação Imperativa;



Programação Funcional x Imperativa

- ✓ Na **programação imperativa**, uma **expressão** é avaliada e seu resultado é armazenado em alguma localização de memória, a qual é representada por uma **variável** (computação feita por sequência de comandos de atribuição);
- ✓ Essa abordagem na **programação imperativa** se concentra no emprego de células de memória, cujos valores representam o **estado** do programa, resultando assim numa metodologia de programação **baixo-nível** (**baixa abstração**);
- ✓ Um programa escrito em **Assembly**, frequentemente deve armazenar também os resultados das avaliações parciais das expressões.
- ✓ Por exemplo: para avaliar $(a + b) / (r - s)$, o valor $(a+b)$ é primeiramente avaliado e armazenado em uma **área intermediária**. Em seguida, $(r - s)$ é avaliado. Para o cálculo final, essas **áreas intermediárias** são usadas, embora os detalhes fiquem escondidos do programador.





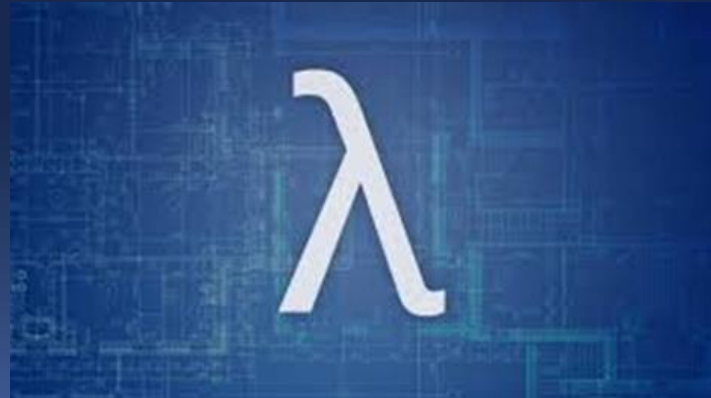
Programação Funcional x Imperativa

- ✓ Uma linguagem de **programação funcional** não usa variáveis ou statements de atribuição, liberando assim o programador de lidar com células de memória e estados de execução;
- ✓ **Sem** variáveis, **construtos iterativos não** são possíveis, uma vez que eles são controlados por **variáveis**;
- ✓ Assim, **repetição** deve ser especificada por meio de **recursão** ao invés de iteração;
- ✓ Programas em linguagens funcionais são **definições** de **funções** e **especificação** de **aplicação** de **funções**;
- ✓ A **execução** desses programas corresponde à **avaliação** da aplicação das funções;
- ✓ A execução de uma função **sempre produz o mesmo resultado** para os mesmos parâmetros de entrada;
- ✓ Essa característica dos programas funcionais é chamada **Transparência Referencial**.



Linguagens Funcionais

- ✓ Provêem um **conjunto** de **funções primitivas**;
- ✓ Provêem um conjunto de **formas funcionais** (Higher Order Functions) para que funções complexas possam construídas a partir das primitivas;
- ✓ Provêem uma operação de **aplicação** da função;
- ✓ Provêem algumas **estruturas** para representar **dados**. Essas estruturas são usadas para representar parâmetros e valores computados pelas funções;
- ✓ Se uma linguagem funcional for bem projetada, ela necessita de **algumas poucas funções primitivas**;



A Linguagem Lisp

- ✓ Primeira Linguagem do Paradigma Funcional;
- ✓ Desenvolvida no **MIT** em 1959;
- ✓ Atualmente existem diversos **dialeto**s do Lisp que incluem algumas características das linguagens **imperativas**;

