

Linguagens e Paradigmas de Programação

**Paradigma Funcional: Linguagens Cat e Joy**

[OUTUBRO 2018]

CONTEÚDO

[1. INTEGRANTES 3](#_Toc528106616)

[2. INTRODUÇÃO 4](#_Toc528106617)

[3. LINGUAGEM FUNCIONAL 5](#_Toc528106618)

[3.1 Definição: 5](#_Toc528106619)

[3.2 Função: 5](#_Toc528106620)

[4. CAT 6](#_Toc528106621)

[4.1 Definição: 6](#_Toc528106622)

[4.2 Função: 6](#_Toc528106623)

[4.3 Exemplo: 6](#_Toc528106624)

[5. JOY 7](#_Toc528106625)

[5.1 Definição: 7](#_Toc528106626)

[5.2 Função: 7](#_Toc528106627)

[5.3 Exemplo: 7](#_Toc528106628)

[6. CONCLUSÃO 9](#_Toc528106629)

[7. BIBLIOGRAFIA 9](#_Toc528106630)

# INTEGRANTES

|  |  |
| --- | --- |
| **Nome** | **R.A.** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# INTRODUÇÃO

# LINGUAGEM FUNCIONAL

## Definição:

Agora, programar em uma linguagem funcional significa basicamente definir funções matemáticas e utilizar o computador para variar essas expressões ou funções.

A maioria das linguagens funcionais são baseadas na notação Lambda de Church, a qual dentre outros detalhes, permite a definição de outras funções, sem a necessidade de especificar um nome, ou seja, definimos apenas a operação que será realizada a partir de uma entrada.

Quanto aos seus fundamentos, há vários pontos que diferem a programação funcional dos paradigmas tradicionais. Primeiro que as linguagens imperativas e orientadas a objetos tem sua implementação baseada na arquitetura de Von Newman, que trabalham basicamente com referência a células de memórias que são as variáveis, e a alteração dos valores contido nas mesmas através de funções, métodos etc.  
  
 Como já dito, o paradigma funcional utiliza a notação de Church, o qual não possui o conceito de referência a célula de memórias, trabalhando apenas com valores passados para funções e produzindo resultado, sendo assim, para determinar a entrada, a função sempre produzirá a mesma saída, evitando os chamados efeitos colaterais.

Outra coisa importante sobre esse paradigma, é que a execução das expressões é controlada por condições e recursão, e não pela sequência ou pela repetição iterativa, o que é comum em linguagem imperativas.

## Função:

As linguagens de programação funcional trazem consigo alto poder de expressividade, com elas, podemos aplicar funções sobre conjuntos de dados, principalmente através de funções de alta ordem, nos permitindo executar computações complexas em chamadas simples de funções.

Sua maior vantagem é sem dúvidas a abstração e também evitar os efeitos colaterais.

# CAT

## Definição:

Cat é uma linguagem de programação funcional, orientada a pilha, inspirada na [linguagem Joy](http://dicionario.sensagent.com/Joy_(linguagem_de_programação)/pt-pt/). A especificação da linguagem está sob [domínio público](http://dicionario.sensagent.com/Domínio_público/pt-pt/), mas sua implementação oficial primária está sob a [licença MIT](http://dicionario.sensagent.com/Licença_MIT/pt-pt/), e foi escrita em [C#](http://dicionario.sensagent.com/C_Sharp/pt-pt/). Há uma segunda implementação oficial, escrita em JavaScript e está sob domínio público.

Joy e Cat diferem da maioria das linguagens funcionais, como Scheme e Heskell, e linguagens formais como o Cálculo Lambda, uma vez que elas se baseiam na composição de funções e não na aplicação de funções.

A linguagem foi criada em 2006 por Chrispher Diggins.

## Função:

A Linguagem CAT foi criada para cumprir as mesmas coisas (Funções) que a linguagem *Joy*, com o diferencial de ser mais restrita e tipada estaticamente, enquanto a *Joy* depende muito da verificação dinâmica. Por consequência, é menos flexível que a linguagem *Joy, porém* mais segura.

## Exemplo:

Neste exemplo simples primeiro inserimos o valor 33, depois inserimos o segundo valor 3 e por último o operando \* para realizar a multiplicação. Na pilha o 33 primeiramente é inserido no topo, logo após o 3 vem acima empurrando o 33 para baixo na pilha, assim como o \* vem por cima novamente. O resultando, 99, fica na posição 0 substituindo os valores digitados anteriormente. Logo após realizamos uma soma entre 1 e 2 e com o resultado multiplicamos por 5, obtendo 15 que fica logo acima na pilha, no exemplo a direita. Em Cat, o valor superior é tratado como o operando direito e o valor abaixo é tratado como o operando esquerdo. Digitamos o sinal maior > para verificar se 99 é maior que 15. O intérprete imprime o resultado, true.

Para limpar a pilha é utilizado o comando pop deixando assim a pilha vazia(empty).  
  
  
 >> 33 3 \*  
 main stack: 99  
 >> 1 2 + 5 \*  
 main stack: 99 15  
 >> >  
 main stack: true  
 >> pop  
 main stack: empty

# JOY

## Definição:

A linguagem Joy é uma linguagem puramente funcional e foi criada em 2001 na Universidade de Melbourne, na Austrália. O criador se chama Manfred von Thun of La Trobe. Joy é baseada na composição de funções assim como o Cálculo Lambda e apresenta muita similaridade com a linguagem Forth. Assim como Factor, é orientada a pilha. É dinâmica e fortemente tipada.

## Função:

Joy é uma linguarem de programação puramente funcional de mais alto nivel, da qual elimina a abstração de Lambda e a Aplicação de Funções e as substitui por cotação de programa e composição de funções.

## Exemplo:

Um exemplo básico de como funciona o Joy:

Para adicionarmos dois integers, falamos 2 e 3, e para escrever a soma deles, você digita o programa:

2 3 +

É assim que ele funciona internamente: o primeiro numeral faz com que o inteiro 2 seja colocado em uma pilha. O segundo numeral faz com que o inteiro 3 seja empurrado em cima disso.

Em seguida, o operador de adição extrai os dois inteiros da pilha e empurra sua soma, 5. Portanto, a notação se parece com o postfix comum. O processador Joy lê programas como os acima até que eles sejam encerrados por um período. Só então eles são executados. No modo padrão, o item no topo da pilha (5 no exemplo) é então gravado no arquivo de saída, que normalmente é a tela

Para calcular o quadrado de um inteiro, ele deve ser multiplicado por si mesmo. Para calcular o quadrado da soma de dois inteiros, a soma deve ser multiplicada por si só. De preferência, isto deve ser feito sem calcular a soma duas vezes. O seguinte é um programa para calcular o quadrado da soma de 2 e 3:

2 3 + dup \*

Após a soma de 2 e 3 ter sido computada, a pilha contém apenas o inteiro 5. O operador *dup* então envia outra cópia do 5 para a pilha.

Então o operador de multiplicação substitui os dois inteiros pelo seu produto, que é o quadrado de 5. O quadrado é então escrito como **25**. Além do operador *dup*, há vários outros para reorganizar o topo da pilha. O operador pop por exemplo, remove o elemento superior, e o operador *swap* troca os dois elementos superiores.

Tudo em Joy é função. Cada função tem uma pilha como argumento e retorna outra pilha como resultado.

DEFINE quadrado == dup \*

Pra vocês verem, em Joy, um número qualquer não é uma constante inteira como em Java, mas sim um programinha que coloca esse número dentro de uma pilha. O operador *\** calcula o produto dos elementos do topo da pilha. Então, a definição (DEFINE) acima de uma função que eleva ao quadrado faz uma cópia do elemento do topo (através do *dup*) e multiplica os dois elementos do topo (a cópia e o elemento que já estava no topo), deixando o quadrado do elemento do topo no topo da pilha – e isso é retornado para o usuário.

# CONCLUSÃO

Por ser uma linguagem 100% funcional e não ter expressões imperativas, é uma linguagem não muito utilizada, pois os conceitos funcionais, que são bons para alguns problemas em específico, são ruins para algumas coisas que são muito normais, tornando assim então o código muito mais complexo para coisas simples.

# BIBLIOGRAFIA