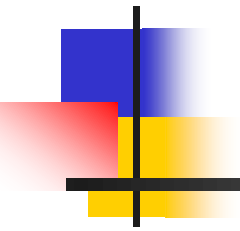
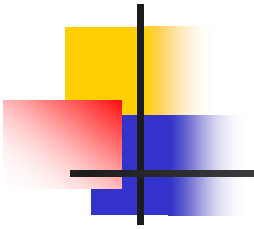


Paradigma Funcional
(visão geral)

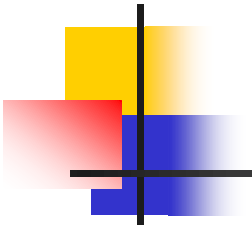
Profª. Rachel Reis

rachel@inf.ufpr.br



Características

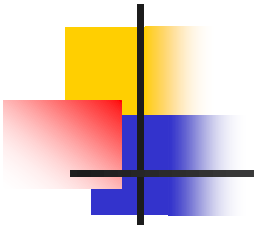
- Em geral, três elementos definem uma linguagem funcional:
  - Uso de funções puras
  - Uso de recursão
  - Avaliação Preguiçosa



Efeito colateral e estados

- Um efeito colateral ocorre quando uma função altera algum estado global do sistema.

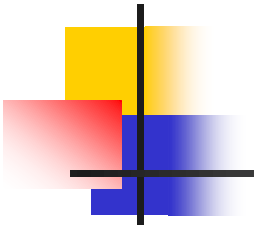
- Exemplo:
  - Alterar uma variável global
  - Ler entrada de dados
  - Imprimir algo na tela



Funções puras

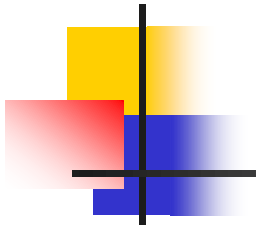
- São funções que não apresentam efeito colateral.
- Ao executar uma função X com a mesma entrada, sempre se obtém a mesma resposta.
- Exemplo:

  Função para somar dois números.



Funções puras

- Quais as vantagens de não se ter efeito colateral?
  - Se o resultado de uma expressão pura não for utilizado, ele não precisa ser calculado.
  - O programa como um todo pode ser reorganizado e otimizado.
  - É possível computar expressões em qualquer ordem (ou até em paralelo).



Pergunta 1

- Função pura só existe se a linguagem for funcional?
  - Resposta: não.

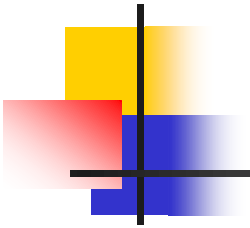
```
int calcularDobro (int num)
{
    return 2 * num;
}
```

Funções puras e impuras

- A função abaixo é pura ou impura?

```
int i = 0;

int calcularDobroMaisI (int num){
    i += 1;
    return 2 * num + i;
}
```

- Resposta: impura, pois ela depende de um estado que não é definido pelos seus parâmetros.



Exercício 1

- Classifique as seguintes funções em C em puras ou impuras:
  - strlen - pura
  - printf - impura
  - getc - impura

Avaliação de Funções

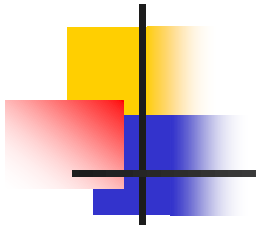
- Função 1: pura ou impura? Impura.

```
void soma_valor(double *soma, int valor)
{
    *soma += valor;
}
```

Avaliação de Funções

- Função 2: pura ou impura? Impura.

```
double calcula_media(int valores[], int n){
    double soma = 0;
    int i;

    for(i = 0; i < n; i++){
        soma_valor(&soma, valores[i]);
    }
    return soma/n;
}
```



Avaliação de Funções

- Funções impuras são virais!
  - Se uma função X chama uma função Y que é impura, então X é impura.

Pergunta 2

- Se uma função X só chama funções puras, ela X é sempre pura?

```
int i = 0;

int calcularDobroMaisI (int num){
      return 2 * num + i;
}
```

- Resposta: não, pois a função X pode depender de algo que não é definido totalmente pelos seus parâmetros.

Pergunta 3

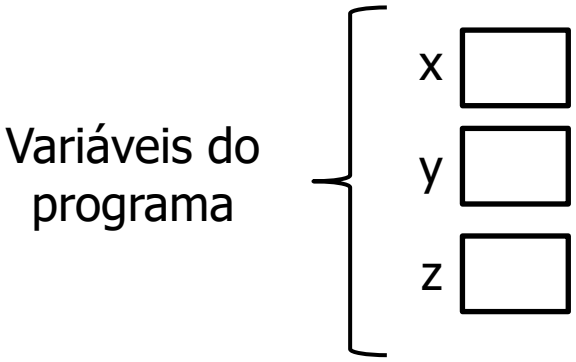
- Um programa que contém apenas funções puras é útil?

- Haskell: deixa as impurezas somente para o ambiente de execução.

Programação sem bugs

- Os estados do programa são fontes de muitos problemas, logo a ausência de estados permite evitar muitos erros de implementação.

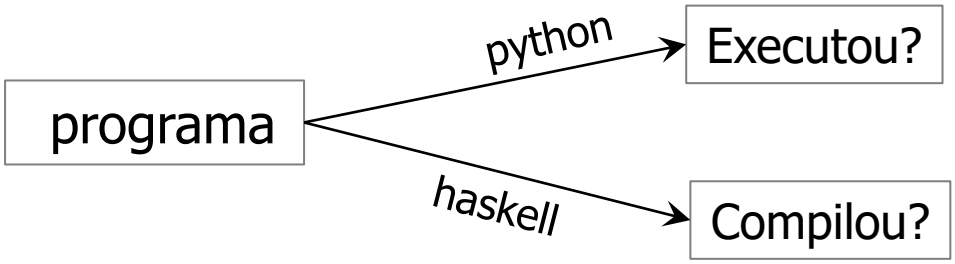
- O que são estados do programa?

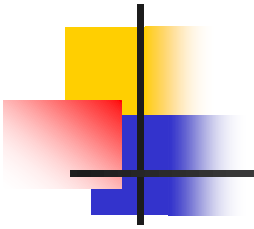


Conteúdo das variáveis em um determinado ponto na execução do programa

Programação sem bugs

- Lema do Haskell: “se compilou, o código está correto!”
- Formas diferentes de se trabalhar com uma linguagem:





Iteração x Recursão

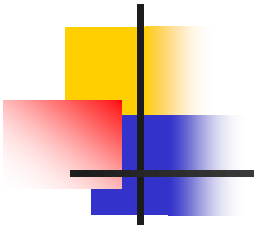
- Em linguagens funcionais, os laços iterativos são implementados via recursão.
  - Como consequência, tem-se um código mais enxuto e declarativo (mostra o que o código faz e não como).

- C/Java

```
int mdc (int a, int b){
    int r = a % b;
    while(r != 0){
        a = b;
        b = r;
        r = a % b;
    }
    return b;
}
```

- Haskell

```
mdc 0 b = b
mdc a 0 = a
mdc a b = mdc b (a `rem` b)
```



Avaliação Preguiçosa

- Algumas linguagens funcionais implementam o conceito de avaliação preguiçosa (*lazy evaluation*).
- Quando uma expressão é gerada, ela gera uma promessa de execução (*thunk*).
- Se em algum momento o valor gerado pela expressão for necessário, o *thunk* é avaliado.

Exemplo em C - Avaliação Estrita

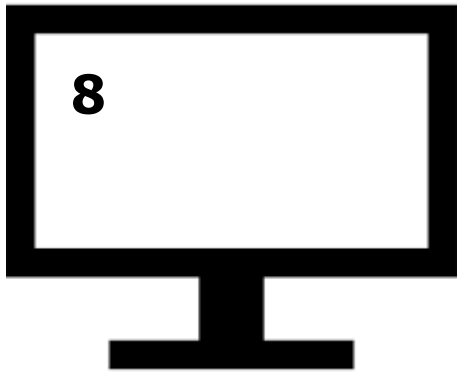
- Avaliação estrita: oposto da avaliação preguiçosa, ou seja, os valores/expressões são sempre avaliados.

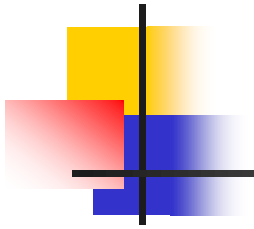
```
int main(){
    int x = 2;
    printf("%d\n", f(x * x, 4 * x + 3));
    return 0;
}

int f(int x, int y){
      return 2 * x;
}
```

Exemplo em C - Avaliação Estrita

- Avaliação estrita: oposto da avaliação preguiçosa, ou seja, os valores/expressões são sempre avaliados.

```
int main(){
    int x = 2;
    printf("%d\n", f(4, 11));
    return 0;
}

int f(int x, int y){
      return 2 * x;
}
```





Avaliação Preguiçosa

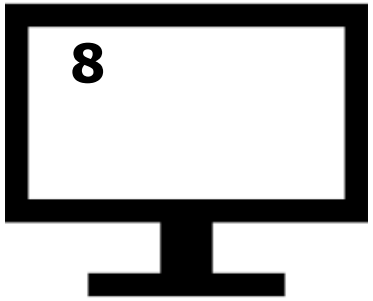
- Exemplo em Haskell.

```
f x y = 2 * x

main = do
    let z = 2
    print (f (z * z) (4 * z + 3))
```

Avaliação Preguiçosa

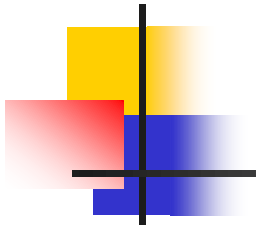
- Exemplo em Haskell.

```
f x y = 2 * x

main = do
    let z = 2
    print (8)
```



- A segunda parte da expressão nunca foi avaliada!

```
print (f (z * z) (4 * z + 3))
```



Avaliação Preguiçosa

- Isso permite a criação de listas infinitas.

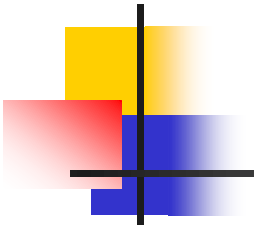
```
[2 * i | i <- [0..]]
```

Paradigma Funcional

- Muitas linguagens de programação estão incorporando elementos do paradigma funcional por conta dos seus benefícios.

- Exemplo Java:

```
public interface List<E>{
    void add(E, x);
    Iterator<E> iteratior();
}
array.stream()
      .filter(n -> (n%2) == 0);
```



Por que Haskell?

- Linguagem puramente funcional (não é multi-paradigma)
- Aceita somente funções puras (como tratar entrada e saída de dados?)
- Declarativa
- Avaliação preguiçosa
- Dados imutáveis
- Tipagem estática e forte

Referências

- Oliveira, A. G. de (2017). Haskell – Uma introdução à programação funcional. Casa do Código.

- Curso de paradigmas de programação (Haskell) da Universidade Federal do ABC (UFABC). Disponível em <https://www.youtube.com/watch?v=eTisiy5FB7k&list=PLYItvall0TqJ25sVTLcMhxsE0Hci58mpQ>. Último acesso em 23/01/2023.