# Fundamentos de Algoritmos e Estrutura de Dados
# Aula 04 – Recursividade e Arvores Binárias

Prof. André Gustavo Hochuli

gustavo.hochuli@pucpr.br
aghochuli@ppgia.pucpr.br

# Plano de Aula

- Discussão Trabalho Hash (Apresentação Grupos)
- Recursão
- Arvores Binárias
  - Codificação Colaborativa: [Link DeepNote]

# Recursividade

- Funções que invocam a si mesma (laço)
  - Critério de Parada
  - Incremento ou Decremento

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print(i)
    print_rec(i-1) #increment/decrement
```

# Recursividade

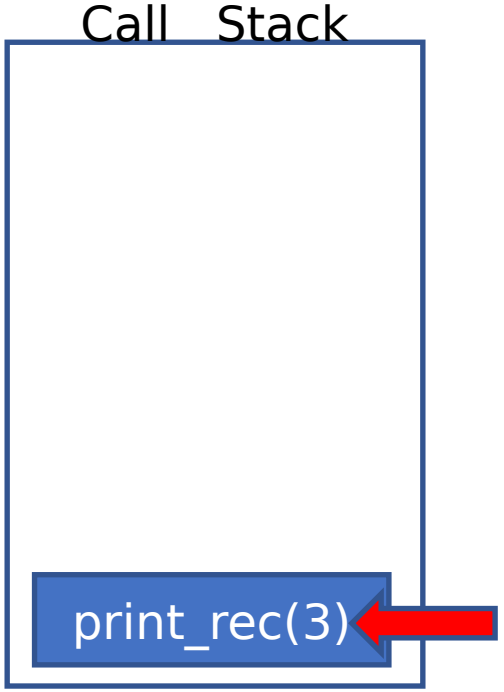
```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print(i)
    print_rec(i-1) #increment/decrement
```

print('main')
print_rec(3)

print('voltei main')

# Recursividade

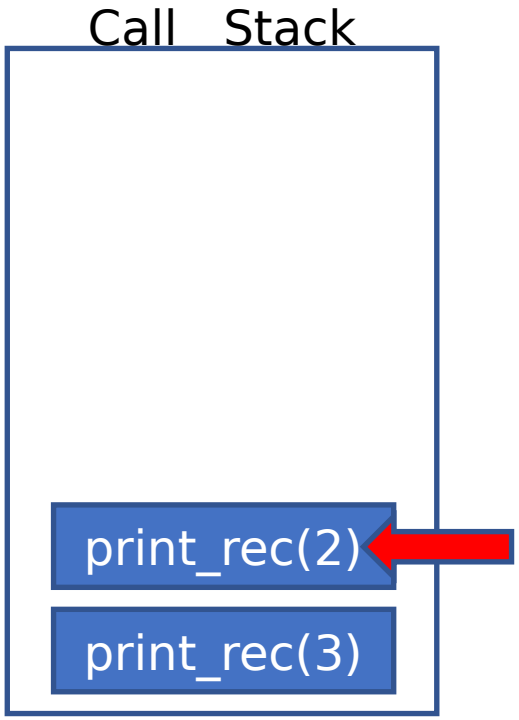
```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

# Recursividade

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print(2)
    print_rec(1) #increment/decrement
```

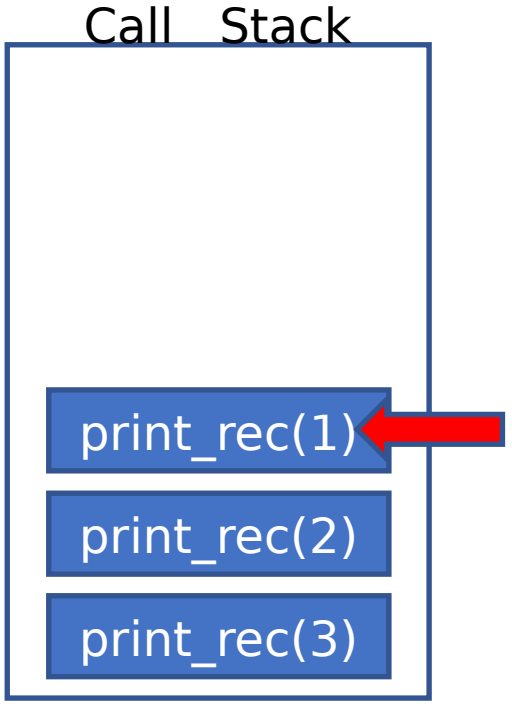



# Recursivid

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print(2)
    print_rec(1) #increment/decrement
```

```
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print(1)
    print_rec(0) #increment/decrement
```

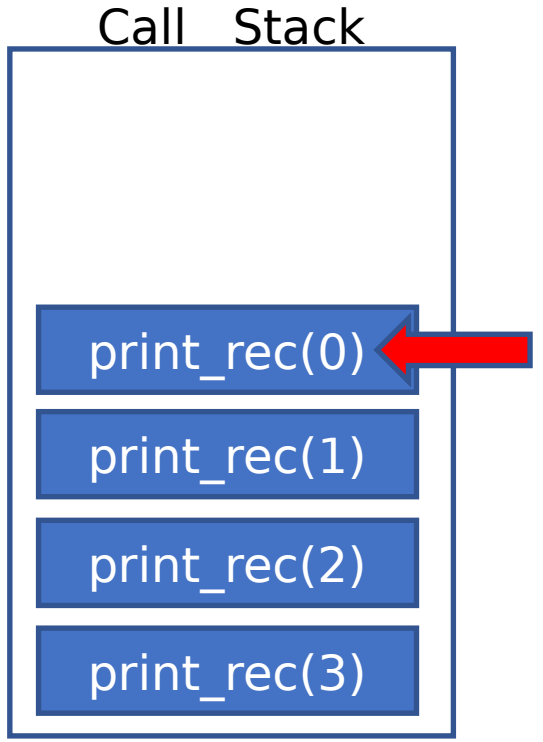
# Recursivid

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print(2)
    print_rec(1) #increment/decrement
```

```
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print(1)
    print_rec(0) #increment/decrement
```

```
def print_rec(0):

    if (0<=0): #Stop Criteria
        return
```

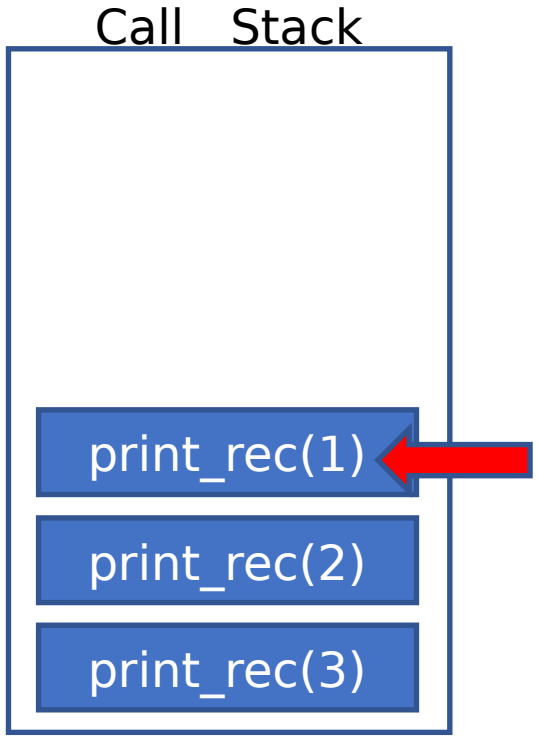



# Recursividade

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print(2)
    print_rec(1) #increment/decrement
```

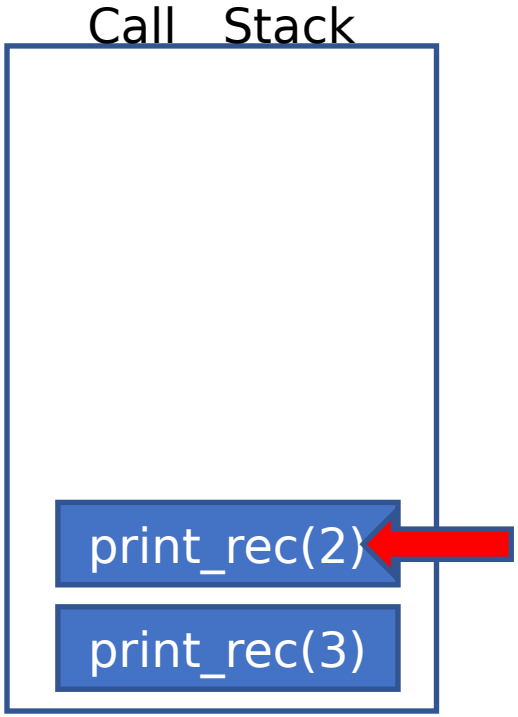
```
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print(1)
    print_rec(0) #increment/decrement
```

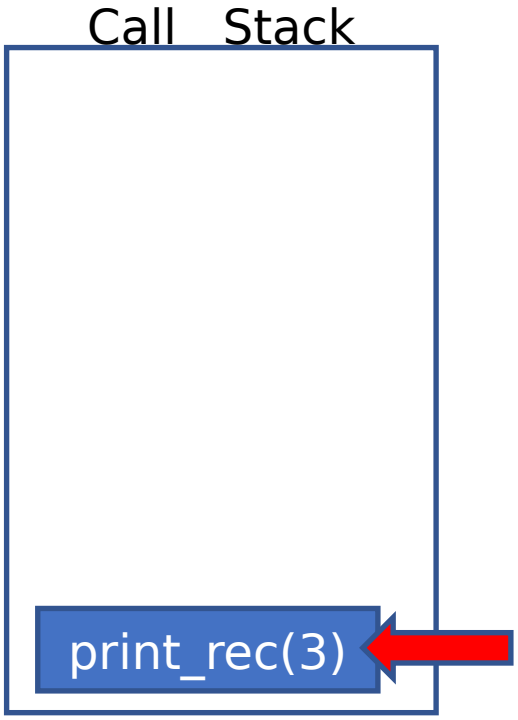
# Recursivid

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print(2)
    print_rec(1) #increment/decrement
```

```python
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print(3)
    print_rec(2) #increment/decrement
```

# Recursividade

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print(i)
    print_rec(i-1) #increment/decrement


print('main')
print_rec(3)
print('voltei main')
```

# Recursividade

- Qual a diferença entre as duas funções abaixo:

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print(i)
    print_rec(i-1) #increment/decrement
```

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print_rec(i-1) #increment/decrement
    print(i)
```

# Recursividade

- Qual a diferença entre as duas funções abaixo:

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print(i)
    print_rec(i-1) #increment/decrement
```

print(i) executa no empilhamento

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print_rec(i-1) #increment/decrement
    print(i)
```

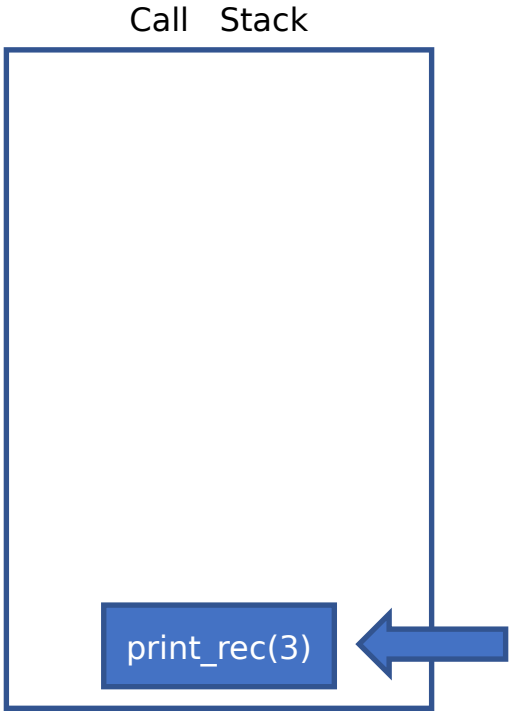
print(i) executa no desempilhamento

# Recursividade

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print_rec(i-1) #increment/decrement
    print(i)


print_rec(3)
```

# Recursividade

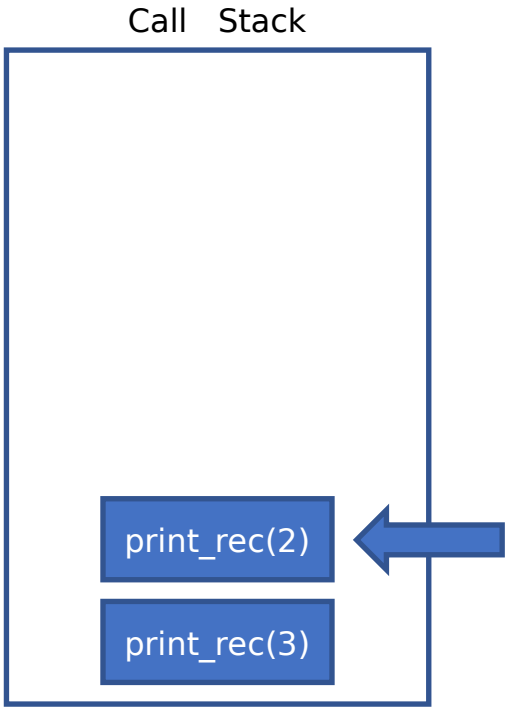
```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```



# Recursividade

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print_rec(1) #increment/decrement
    print(2)
```

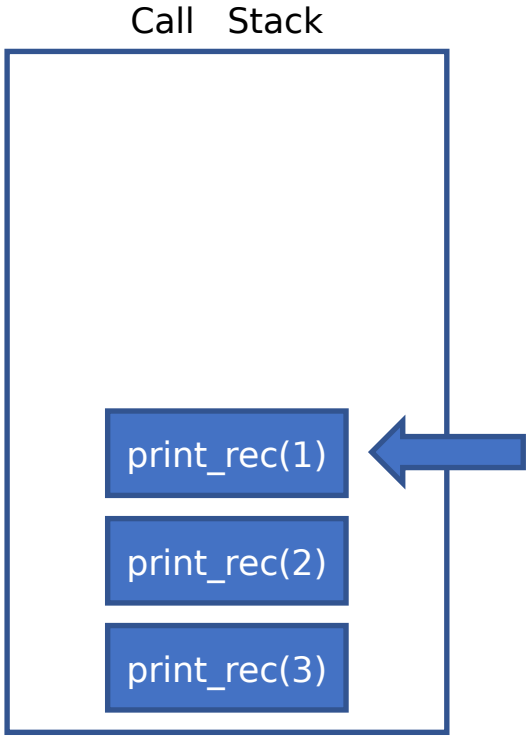


# Recursividade

```python
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

```python
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print_rec(1) #increment/decrement
    print(2)
```

```python
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print_rec(0) #increment/decrement
    print(1)
```



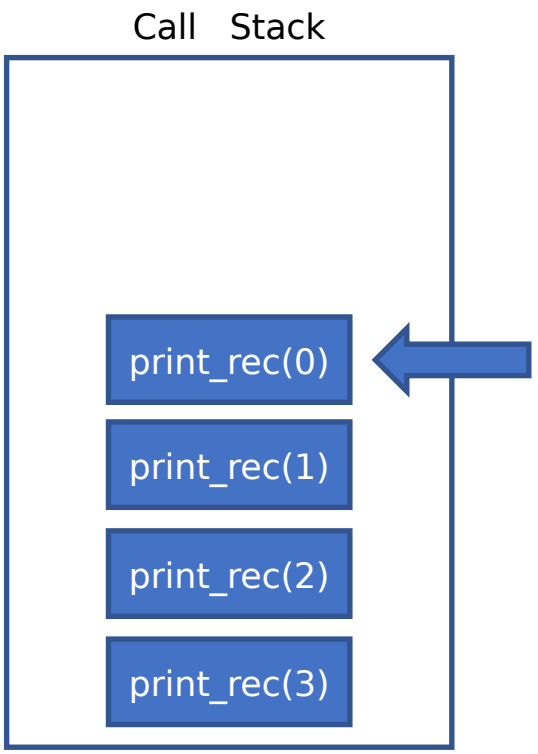
# Recursividade

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print_rec(1) #increment/decrement
    print(2)
```

```
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print_rec(0) #increment/decrement
    print(1)
```

```
def print_rec(0):

    if (0<=0): #Stop Criteria
        return
```

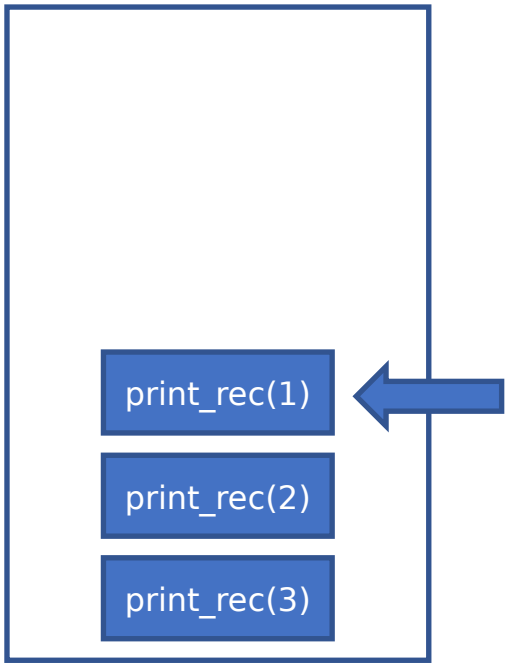


# Recursividade

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print_rec(1) #increment/decrement
    print(2)
```

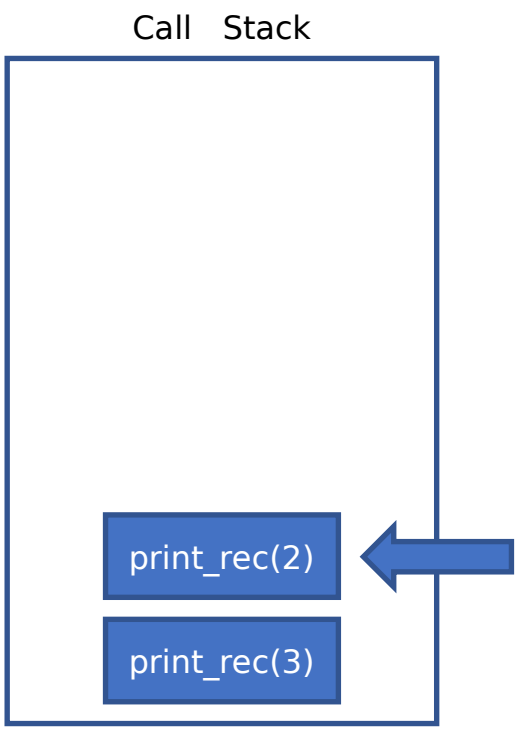
```
def print_rec(1):

    if (1<=0): #Stop Criteria
        return

    print_rec(0) #increment/decrement
    print(1)
```

# Recursividade

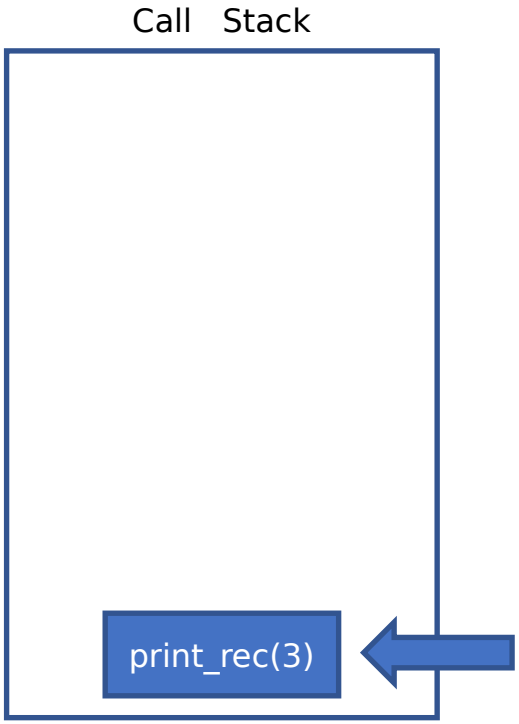
```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

```
def print_rec(2):

    if (2<=0): #Stop Criteria
        return

    print_rec(1) #increment/decrement
    print(2)
```

# Recursividade

```
def print_rec(3):

    if (3<=0): #Stop Criteria
        return

    print_rec(2) #increment/decrement
    print(3)
```

# Recursividade

```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print_rec(i-1) #increment/decrement
    print(i)


print_rec(3)
```

# Recursividade

- Implemente uma soma recursiva de 0....N
  - N = 3 | Soma = 3 + 2 + 1 ou 1 + 2 + 3
  - N = 3 | Soma = 5 + 4 + 3 + 2 + 1 ou 1 + 2 + 3 + 4 + 5

- Implemente o Fibonacci Iterativo

# Recursão vs Iteração

- Cada situação demanda uma abordagem

- Via de regra, a recursão apresenta *overhead* em relação a iterativa
  - No entanto, veremos que em algumas situações a aplicação de recursão é sugestiva

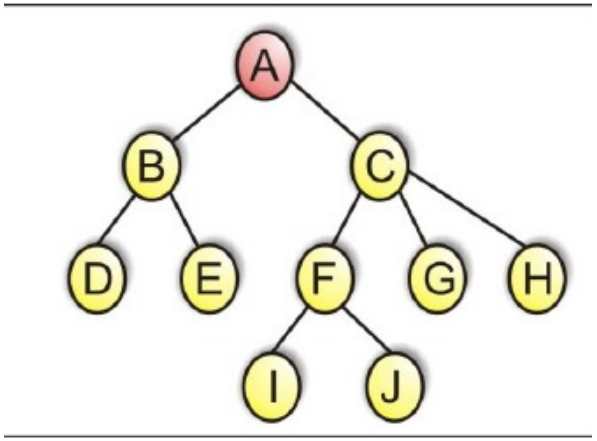
```
def print_rec(i):

    if (i<=0): #Stop Criteria
        return

    print_rec(i-1) #increment/decrement
    print(i)
```

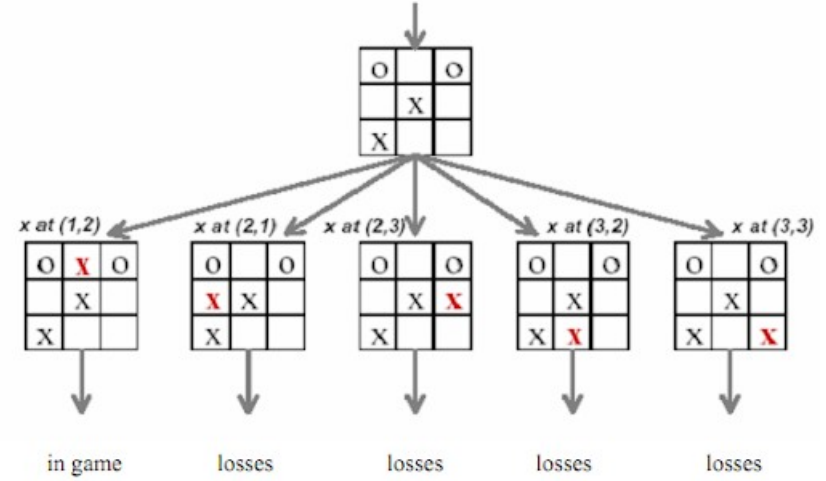
```
def print_iter(i):

    for n in range(i)
        print(n)
```

# Árvores

# Árvores

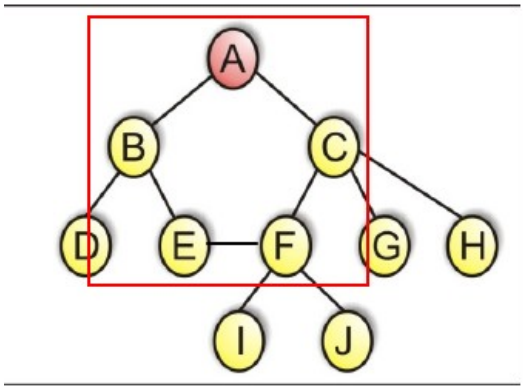
- **Estrutura não linear**
- **Representação Hierárquica**
- **Aplicações**
  - **Verificadores de sintaxe**
  - **Banco de Dados**
  - **Roteadores**
  - **Escalonadores de processos**
  - **I.A**



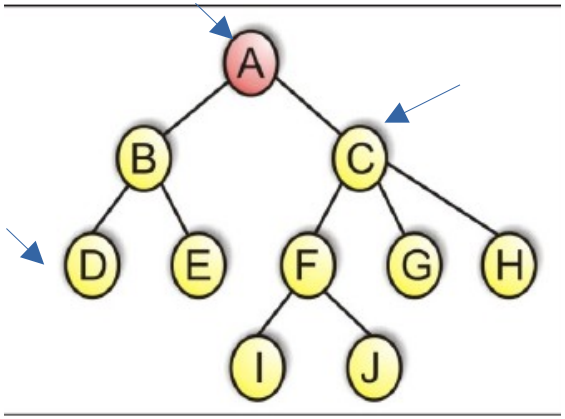



# Árvores - Conceitos

- **Árvore não contém ciclos**

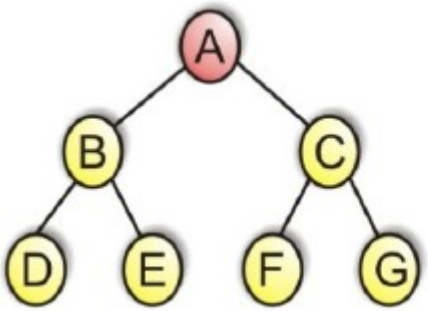


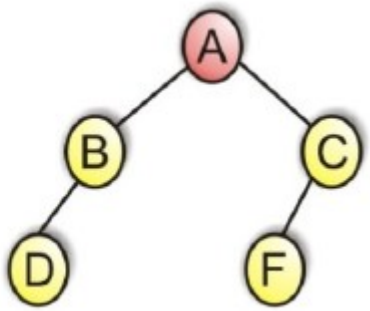

- **Grau:**
  - **Número de sub-árvores**
    - **Nós: A=2, C=3, D=0**
    - **Grau Árvore: 3**
- **Nível**
  - **Distância entre o vértice até a raiz**
    - **Nós: D=2, I=3**
    - **Nível da Árvore: 3**



# Árvores - Binárias

- **Árvores Binárias tem grau 2**

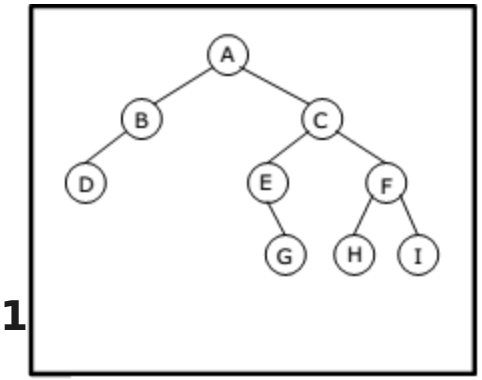



- **Caminhamento**
  - **Pré-Ordem: raiz→esq→dir**
  - **Pós-Ordem: esq →dir→raiz**
  - **In-Ordem: esq→raiz→dir**
  - **Nível*: raizes(N=0)→raizes(N=1**

    (*) Método não recursivo

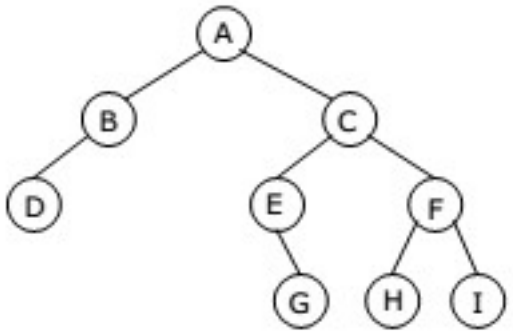
    (*) Árvore deve ser convertido em fila




- Preorder
  A, B, D, C, E, G, F, H, I
- Postorder
  D, B, G, E, H, I, F, C, A
- Inorder
  D, B, A, E, G, C, H, F, I
- Level order
  A, B, C, D, E, F, G, H, I

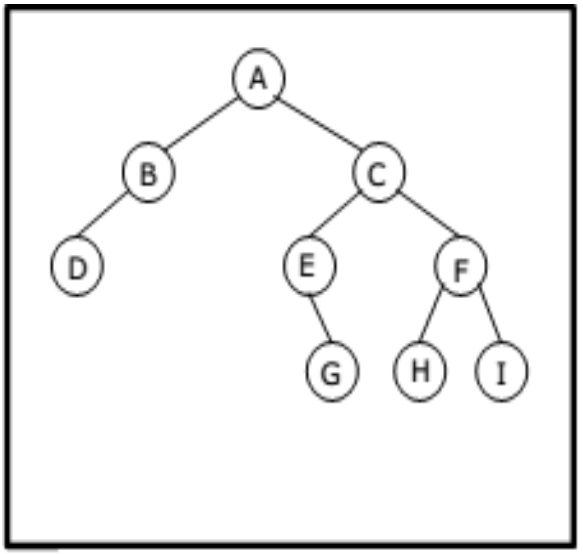
# Árvores - Implementação

```
class Node:
    def __init__(self,data):
        self.data = data  # Assign data
        self.left = None  # Initialize as None
        self.right = None  # Initialize as None

class Binary_Tree:

    # Init Class
    def __init__(self,data):
        self.root = Node(data)

    def push(self,data):

        if self.root is None:
            print("Root")
            self.root = Binary_Tree(data)

        if data > self.root.data:
            if self.root.right is None:
                print("Add Right")
                self.root.right = Binary_Tree(data)
            else:
                self.root.right.push(data)

        else:
            if self.root.left is None:
                print("Add Left")
                self.root.left = Binary_Tree(data)
            else:
                self.root.left.push(data)

        return
```



# Árvores - Caminhamento

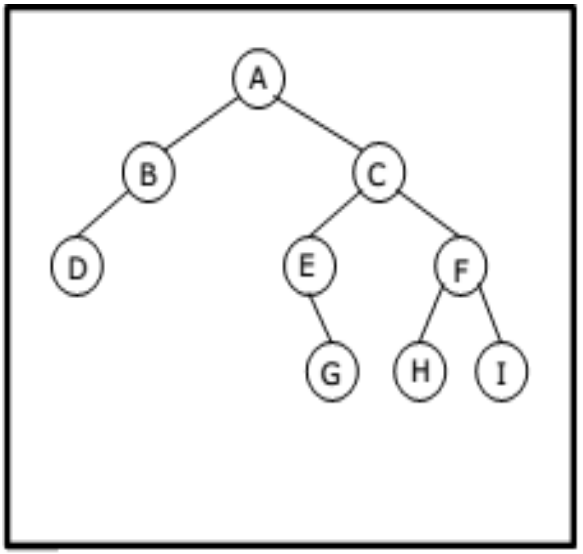
```
def walk_preorder(self):

    print(self.root.data)

    if self.root.left is not None:
        self.root.left.walk_preorder()

    if self.root.right is not None:
        self.root.right.walk_preorder()

def walk_inorder(self):

    if self.root.left is not None:
        self.root.left.walk_inorder()

    print(self.root.data)

    if self.root.right is not None:
        self.root.right.walk_inorder()


def walk_postorder(self):

    if self.root.left is not None:
        self.root.left.walk_postorder()


    if self.root.right is not None:
        self.root.right.walk_postorder()

    print(self.root.data)
```




- Preorder
  A, B, D, C, E, G, F, H, I
- Postorder
  D, B, G, E, H, I, F, C, A
- Inorder
  D, B, A, E, G, C, H, F, I
- Level order
  A, B, C, D, E, F, G, H, I

# Árvores - Caminhamento

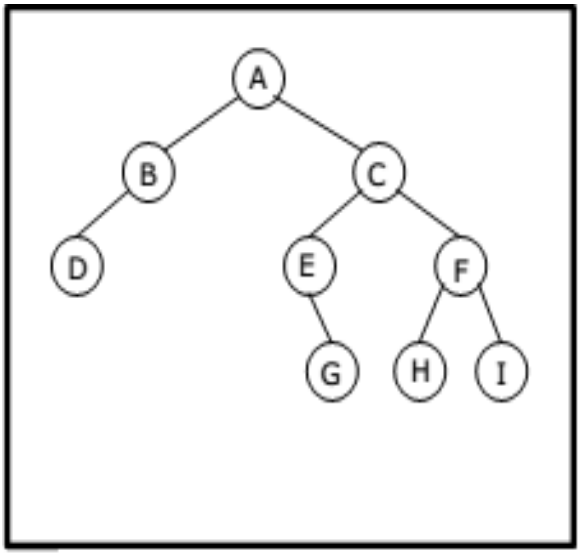
```
def walk_preorder(self):

    print(self.root.data)

    if self.root.left is not None:
        self.root.left.walk_preorder()

    if self.root.right is not None:
        self.root.right.walk_preorder()
```




- Preorder
  A, B, D, C, E, G, F, H, I
- Postorder
  D, B, G, E, H, I, F, C, A
- Inorder
  D, B, A, E, G, C, H, F, I
- Level order
  A, B, C, D, E, F, G, H, I

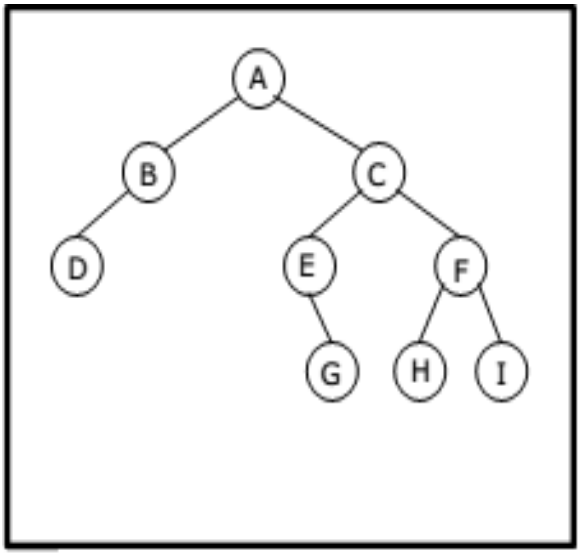
# Árvores - Caminhamento

```
def walk_inorder(self):

    if self.root.left is not None:
        self.root.left.walk_inorder()

    print(self.root.data)

    if self.root.right is not None:
        self.root.right.walk_inorder()
```




- Preorder
  A, B, D, C, E, G, F, H, I
- Postorder
  D, B, G, E, H, I, F, C, A
- Inorder
  D, B, A, E, G, C, H, F, I
- Level order
  A, B, C, D, E, F, G, H, I

# Árvores - Caminhamento




- Preorder
  A, B, D, C, E, G, F, H, I
- Postorder
  D, B, G, E, H, I, F, C, A
- Inorder
  D, B, A, E, G, C, H, F, I
- Level order
  A, B, C, D, E, F, G, H, I

```
def walk_postorder(self):

    if self.root.left is not None:
        self.root.left.walk_postorder()


    if self.root.right is not None:
        self.root.right.walk_postorder()

    print(self.root.data)
```

# Árvores - Exercício

**Implementação de um tradutor de código morse**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | ·− | J | ·−−− | S | ··· | 2 | ··−−− |
| B | −··· | K | −·− | T | − | 3 | ···−− |
| C | −·−· | L | ·−·· | U | ··− | 4 | ····− |
| D | −·· | M | −− | V | ···− | 5 | ····· |
| E | · | N | −· | W | ·−− | 6 | −···· |
| F | ··−· | O | −−− | X | −··− | 7 | −−··· |
| G | −−· | P | ·−−· | Y | −·−− | 8 | −−−·· |
| H | ···· | Q | −−·− | Z | −−·· | 9 | −−−−· |
| I | ·· | R | ·−· | 1 | ·−−−− | 0 | −−−−− |

--- .-.. .-/-- ..- -. -.. --- = Ola Mundo