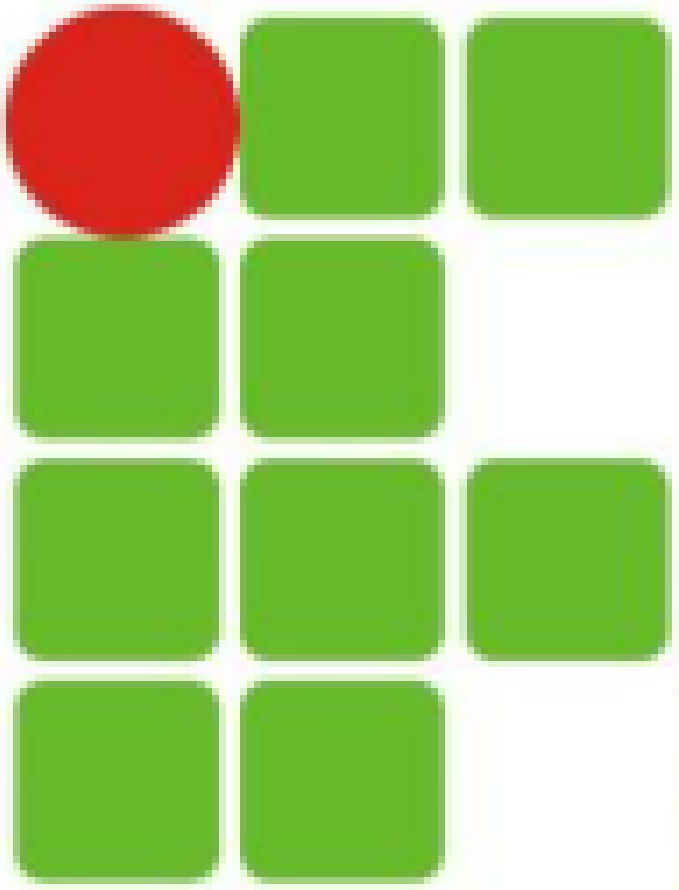
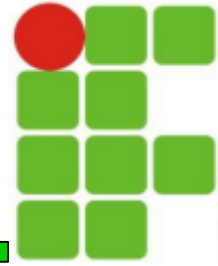


Python: Funções

Galileu Batista de Sousa

Galileu.batista -at +ifrn -edu +br

Fundamentos de Funções

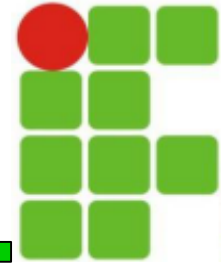


- Funções são **trechos de código reutilizáveis**
- Já usamos várias funções:

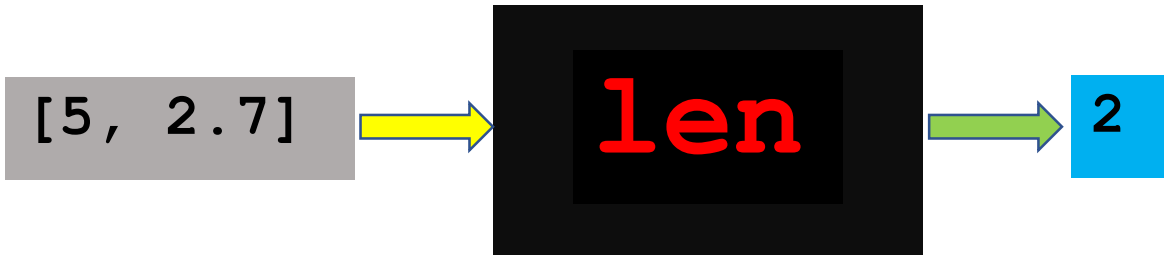
```
filhos   = ["Tiago", "Amanda", "Giovanna" ]
tam = len (filhos)
print ("O # de filhos é: ", tam)
```

- Essas funções estavam "prontas"
  - São ditas funções "built-in"
- Podemos criar nossas próprias funções

Por que usar funções?

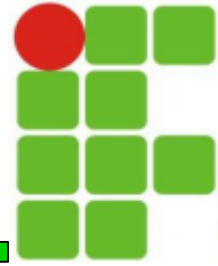


- **Reutilizar código** - obviamente
- Talvez a principal motivação seja:
  - Controlar a complexidade
    - Desenvolve-se uma lógica complexa
    - Encapsula-se dentro de uma função
    - Esquece como ela foi feita – apenas usa o código.




```
def len(conj):
    tam = 0
    for e in conj:
        tam += 1
    return tam
```

Matemática x Programação



- Na matemática um função é:
  - Um conjunto de operações matemáticas
  - Aceita elementos de um conjunto origem (domínio)
  - Retorna elementos em um conjunto destino (contra domínio)
- Exemplo:

f(x) : 2x

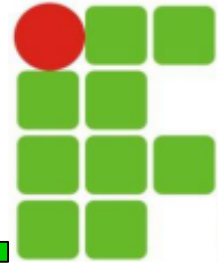
f(2) => 4

f(10) => 20

f(f(5)) => 20

- Qual o nome dessa função?
- O fato dessa função existir:
  - Obriga você conhecê-la?
  - Obriga você a usá-la?

Matemática x **Programação**

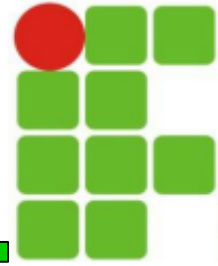


- Na programação um função é:
  - Um conjunto de comandos
  - Aceita valores de entrada (argumentos ou parâmetros)
  - Retorna elementos de saída (retorno)
- Exemplo:

```
def f(x):
    y = 2 * x
    return y

print (f(2))
print (f(f(5))
```

- Qual o nome dessa função?
- O fato dessa função existir:
  - Obriga você conhecê-la?
  - Obriga você a usá-la?

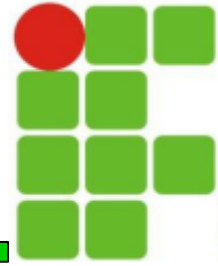
Quando criar/usar funções?



- Quando observar:
  - que seu programa está grande
    - Coloque um trecho numa função
  - uma funcionalidade padrão
    - Calcular o fatorial de um número
  - Que a solução pode ser decomposta em partes
    - Estratégia de ***divisão e conquista***
    - Um jogo tem:
      - A parte da estratégia
      - A parte visual

Crie uma ou mais funções
Chame a função onde estava o trecho original

Funções na prática

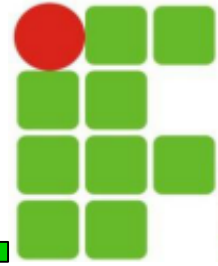


- Programa para escrever os fatoriais dos números 1 a 100
  - Esse é um problema que envolve
    - Laços encadeados. Por que?
  - E se já existisse uma função que calculasse o fatorial?

```
for n in range (1, 101):
    print (n, fatorial(n))
```

  - Mas "não existe" essa função fatorial.
    - O que fazer?
    - Criar, nós mesmos, a função fatorial

A função fatorial

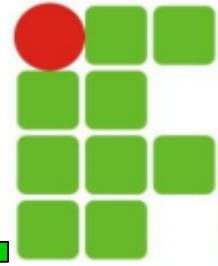


- O que deve fazer a função fatorial?
  - Receber um valor
  - Entregar como resultado o fatorial desse valor

```
def fatorial (n):
    fat = 1
    for i in range (1, n+1):
        fat *= i
    return fat
```

  - Essa trecho de código (função) não é executado por si só
    - Nem sabe se será ou quando será

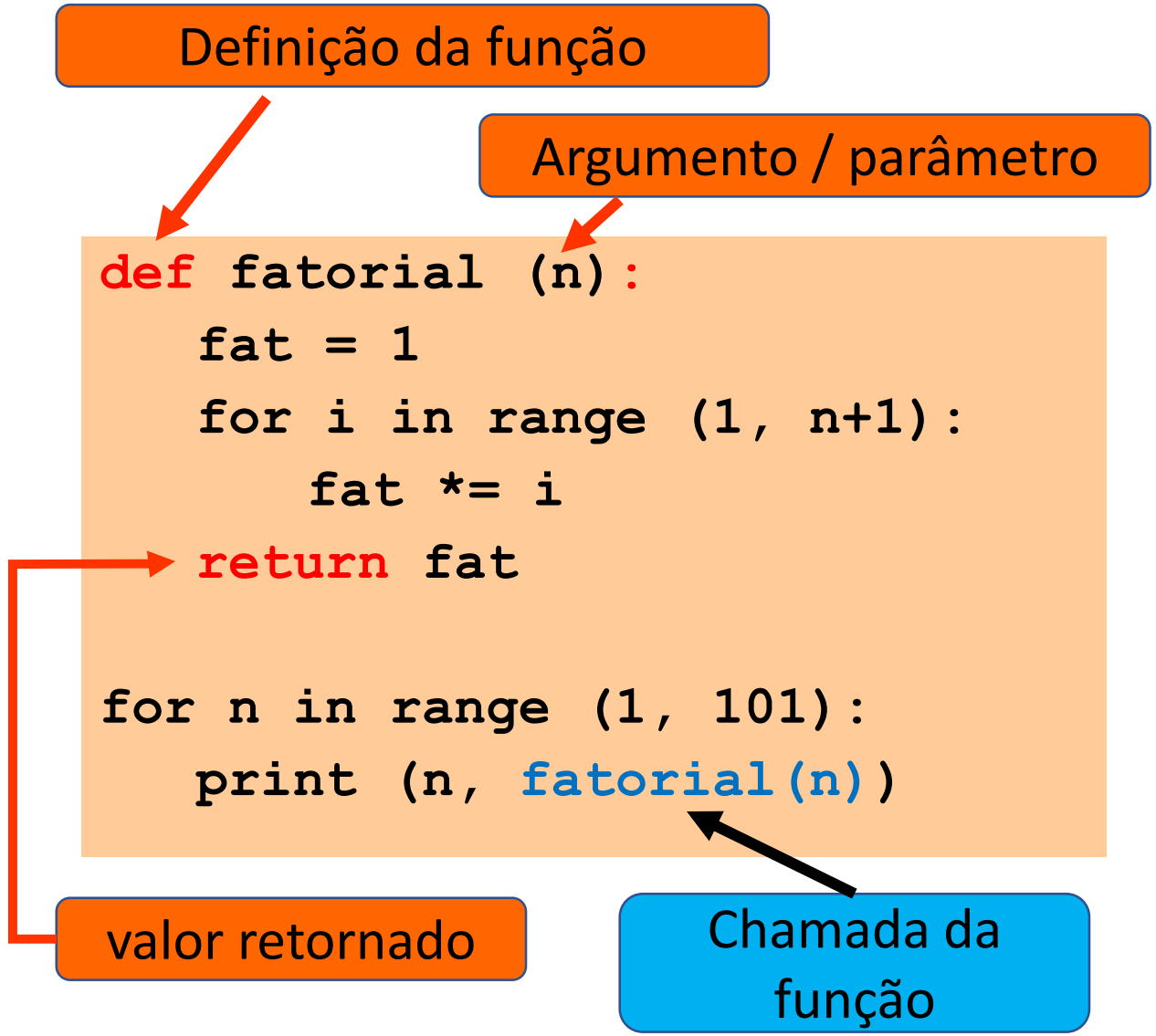
O programa completo



```
def fatorial (n):
    fat = 1
    for i in range (1, n+1):
        fat *= i
    return fat

for x in range (1, 101):
    print (n, fatorial(x))
```

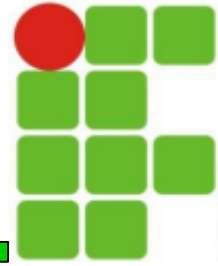
- O código da função:
  - Não é executado assim que encontrado
  - Somente quando chamado
- A expressão no momento da chamada é copiada para o argumento
- As variáveis internas da função somente são vistos nela

O programa completo



- O código da função:
  - Não é executado assim que encontrado
  - Somente quando chamado
- A expressão no momento da chamada é copiada para o argumento
- As variáveis internas da função somente são vistos nela

Detalhes sobre argumentos

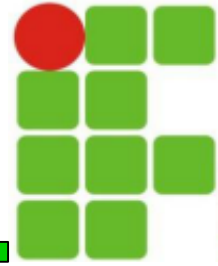


Argumentos / parâmetros

```
def maior (a, b):
    if a > b:
        maior = a
    else:
        maior = b
    return maior

x = int (input("Valor 1: ")
y = int (input("Valor 2: ")

print (maior(2*x, y))
```

- Podem existir vários argumentos
- Na chamada, as expressões são avaliadas antes; os argumentos já recebem os valores
- Mudanças de valores nos argumentos dentro da função não alteram valores externos

Detalhes sobre retorno

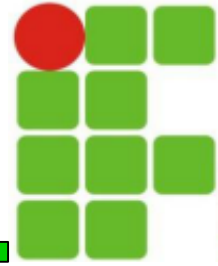


Argumentos / parâmetros

```
def maior (a, b):
    if a > b:
        return a
    else:
        return b

x = int (input("Valor 1: ")
y = int (input("Valor 2: ")

print (maior(2*x, y))
```

- Podem existir vários comandos return
  - O primeiro que for atingido provoca o final da função
- Não é necessário ter comando return – a função termina no final do código

Detalhes sobre chamada



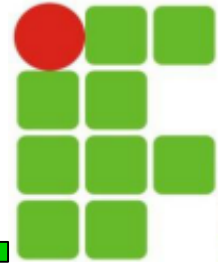
Argumentos / parâmetros

```
def maior (a, b):
    if a > b:
        return a
    else:
        return b

x = int (input("Valor 1: ")

print (maior(b=5, a=x))
```

- É possível nomear a qual parâmetro cada valor na chamada será atribuído
- Não necessita respeitar a ordem de definição na função

O argumento b terá o valor 5, enquanto a receberá o que foi lido

Detalhes sobre variáveis locais



x sendo usado na função

```
def maiorQueX (a):
    if a > x:
        return True
    else:
        return False

x = int (input("Valor 1: ")

print (maiorQueX(10))
```

- As variáveis de uma função são internas a ela.
  - Há possibilidade de funções dentro de funções, mas ....
- Uma função **pode acessar as variáveis globais**
  - **Mas não modificar**
    - Vira variável local