Roteiro

- Funções
- Condicionais
- Iteração

Funções

$f(x) = 2 * x$
$f(3) \rightarrow 2 * 3 \rightarrow 6$

$f(x) = x^2$
$f(4) \rightarrow 4^2 \rightarrow 16$

$f(x, y) = (x + y) * 2$
$f(2,3) \rightarrow (2 + 3) * 2 \rightarrow 10$

Funções matemáticas

- Exemplo de chamada de função:

```
>>> type('32')
<type 'str'>
```

  - A função *type* exibe o tipo da variável
- Funções são estruturas que recebem valores e retornam um resultado

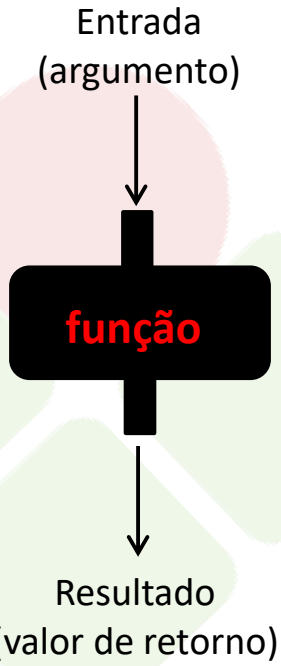
Funções

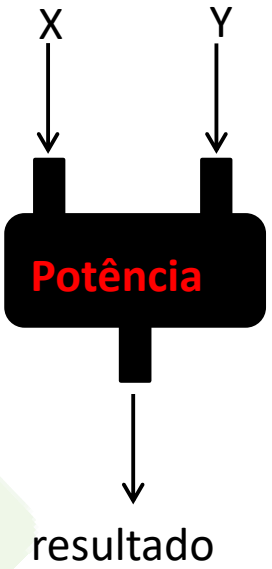
- Argumento
  - Valores recebidos pela função
- Valor de retorno
  - Resultado da função
  -
- Funções retornam resultados que podem ser armazenados em variáveis

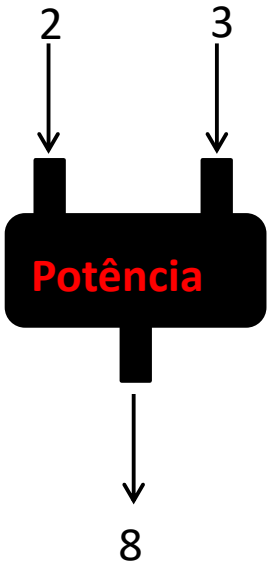
```
>>> a = type('32')
>>> print a
<type 'str'>
```

```
>>> n = len('Hello')
>>> print n
5
```

Funções


Entrada
(argumento)
função
Resultado
(valor de retorno)


X
Y
Potência
resultado


2
3
Potência
8

Funções

- Conversão de tipos

```
>>> int('32')
32
>>> int('Hello')
ValueError: invalid literal for int() with base 10: 'Hello'
```

```
>>> int(3.14159)
3
>>> int(-2.3)
-2
```

```
>>> float('3.14159')
3.14159
>>> float(32)
32.0
```

```
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

Funções

- Coerção entre tipos
  - Divisão de números inteiros

```
>>> 50 / 60
0
>>> 10 / 3
3
```

  - Se um dos operandos for do tipo *float*, é feita a coerção do resultado para *float*

```
>>> 50.0 / 60
0.8333333333333334
>>> 10 / 3.0
3.3333333333333335
```

Funções

- Funções matemáticas
  - sen(pi/2)
  - cos(1/2)
- Python fornece essas funções já prontas
  - Módulo matemático → **math**
    - Módulo é um arquivo que contém funções já implementadas para serem usadas pelo programador
    - Evita diariamente a reinvenção da roda
    - Antes de usar as funções prontas, temos que importar o módulo

```
>>> import math
```

Funções

- Funções matemáticas
  - Chamada de funções: notação de ponto
    - nome_módulo.nome_função()

```
>>> import math
>>> print math.factorial(3)
6
>>> print math.sqrt(9)
3
>>> print math.factorial(4)
24
>>> x = math.factorial(math.sqrt(25))
>>> print x
120
```

  - Qual a importância de termos módulos prontos disponíveis?

Funções

- Como descobrir se determinada funcionalidade já existe implementada por algum módulo?
  - Experiência
  - Pesquisa

```
>>> import nome_do_módulo
>>> dir(nome_do_módulo)
Aqui deve aparecer uma série de nomes de funções
que estão disponíveis no módulo chamado nome_do_módulo.
```

```
>>> import math
>>> dir(math)
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh',
'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees',
'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
```

Funções

- Definindo nossas próprias funções
  - Até agora, usamos funções definidas (criadas) por outras pessoas
  - Vamos criar uma função que imprima o dobro do número 3?

Definição de função

```
def dobro_de_tres():
    print 2 * 3


dobro_de_tres()
```

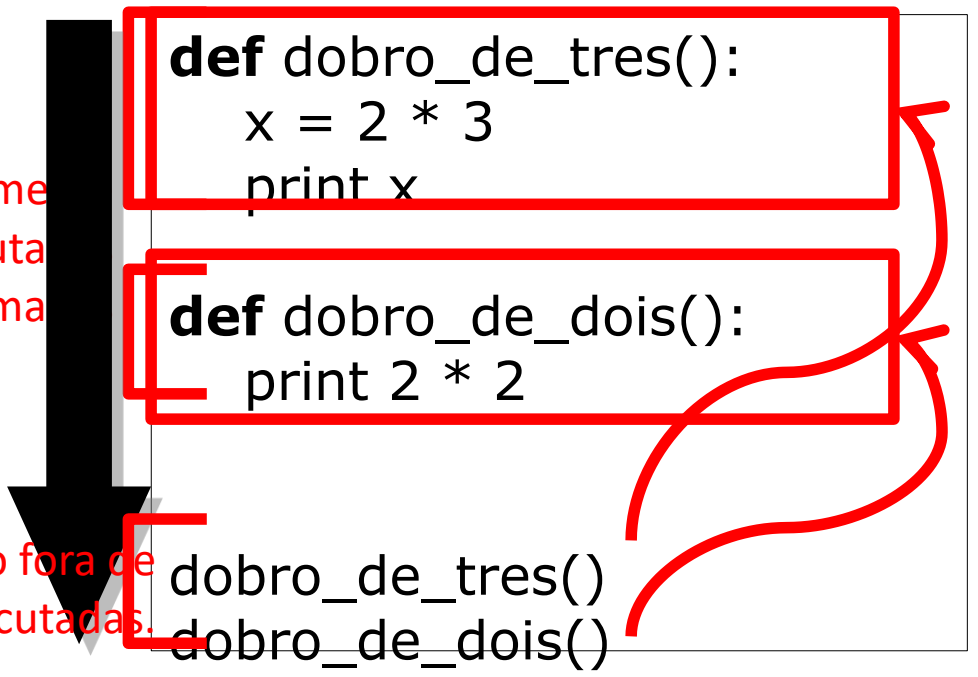
Chamada de função

```
O dobro de 3 é 6
```

Fluxo de Execução de um Programa

Funções somente são executadas quando chamadas

Somente as linhas de código fora de funções é que são executadas.



Parâmetros e Argumentos

- Entradas da função
  - Se quisermos calcular a raiz quadrada de um determinado número, precisamos indicar que número é.

```
import math
print math.sqrt(16)
```

  - Função que imprime o dobro de um número

```
def dobra(num):
    print 2 * num
```

    - Recebe um argumento e atribui ao parâmetro **num**

```
>>> dobra(10)
20
>>> dobra(7)
14
```

```
>>> dobra(4 * 2)
16
>>> dobra(32 / 2)
32
```

Parâmetros e Argumentos

```
def dobra(num):
    print 2 * num
```

```
>>> z = 100
>>> dobra(z)
200
>>> minha_var = 4
>>> dobra(minha_var)
8
```

```
def juntaPalavras(p1, p2):
        juncao = p1 + p2
        print juncao
```

```
>>> juntaPalavras('Ola', ' mundo!')
Ola mundo!

>>> print p1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'p1' is not defined

>>> print juncao
```

Exercícios

1. Escreva uma função que receba dois números como entrada e imprima a soma dos dois.
2. Escreva uma função que receba um número como entrada e imprima na tela o valor desse número elevado ao cubo.
3. Escreva uma função que receba como entrada um número e imprima na tela somente o último dígito desse número:
   - 546 → 6
   - 449 → 9
4. Escreva uma função que receba como entrada dois números representando horas e minutos, respectivamente, e mostre na tela as horas no formato hh:mm
   - 12 e 34 → 12:34 ; 22 e 12 → 22:12

Roteiro

- ~~Funções~~ (por enquanto)
- **Condicionais**
- Recursividade
- Iteração

Expressões Booleanas

- Pode assumir dois valores
  - True
  - False

```
>>> 5 == 5
True
>>> 5 == 4
False
>>> 5 != 6
True
>>> 5 != 5
False
>>> 5 < 6
True
>>> 5 > 6
False
>>> 5 <= 5
True
>>> 5 >= 6
False
```

Execução Condicional

- Na maioria dos programas é necessário checar determinadas condições para definir qual caminho tomar
  - Ex.:
    - Se o usuário fornecer a senha correta, siga para a tela de cadastro, caso contrário, mostre a tela de erro.
    - Se o valor a ser debitado for maior que o saldo, mostre uma mensagem de erro, caso contrário, faça o débito.

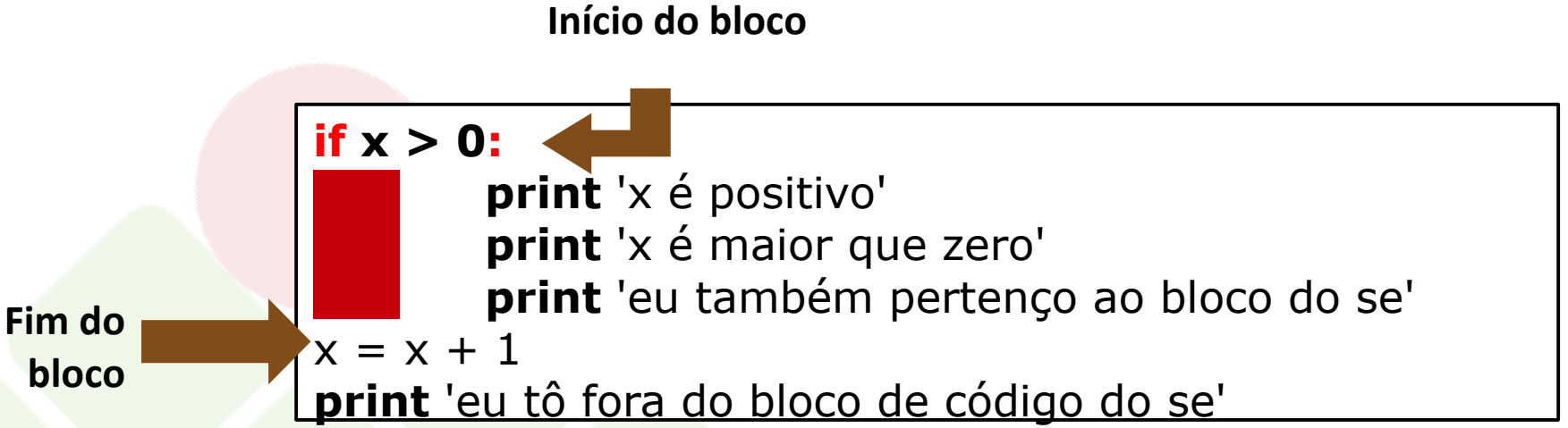
Se

```
x = input('Digite um valor positivo: ')

if x < 0:
        print 'o valor fornecido é negativo'
```

Execução Condicional

- Bloco de código
  - Delimita qual código será executado quando a condição do **if** for verdadeira



  - Em Python, o que delimita o bloco de código é a indentação do código

Execução Condicional

- Exercício
  - Escreva um programa Python que solicite um número ao usuário e escreva uma mensagem na tela caso o número fornecido for par.

```
x = input('Digite um número: ')
if x % 2 == 0:
        print 'O número é par!'
```

  - Escreva um programa que verifique se dois números fornecidos pelo usuário são múltiplos.

Execução Alternativa

- Se a condição for verdadeira, executa algo, mas e se não for, o que o programa faz?
  - Instrução **else** (senão)

Se o resto da divisão de x por 2 for 0

Senão

```
1:      x = input('Digite um número: ')
2:      if x % 2 == 0:
3:          print 'O número é par'
4:      else:
5:          print 'O número é ímpar'
```

  - Se a condição for verdadeira, executa a linha 3
  - Se a condição for falsa, executa a linha 5

Execução Alternativa

- Exercício
  - Faça um programa que leia um número e diga se esse número é positivo ou negativo.

```
x = input('Digite um número: ')
if x > 0:
        print 'O número é positivo!'
else:
        print 'O número é negativo!'
```

Errado! E se for 0?

Senão se

```
x = input('Digite um número: ')
if x > 0:
        print 'O número é positivo!'
elif x < 0:
        print 'O número é negativo!'
else:
        print 'O número é zero!'
```

Condicionais Encadeados

- Comparar dois números, **x** e **y**

```
x = input('Digite um número: ')
y = input('Digite outro número: ')
if x > y:
        print 'O primeiro é maior que o segundo!'
else:
        print 'O segundo é maior que o primeiro!'
```

Errado! E se forem iguais?

```
x = input('Digite um número: ')
y = input('Digite outro número: ')
if x > y:
        print 'O primeiro é maior que o segundo!'
elif x < y:
        print 'O segundo é maior que o primeiro!'
else:
        print 'Os dois números são iguais!'
```

Condicionais Aninhados

```
x = input('Digite um número: ')
y = input('Digite outro número: ')
if x > y:
        print 'O primeiro é maior que o segundo!'
else
        if x < y:
                print 'O segundo é maior que o
primeiro!'
        else:
                print 'Os dois números são iguais!'
```

Condicionais Aninhados

- ifs aninhados...

```
x = input('Digite um número: ')
if x > 0:
  if x < 10:
    print 'x é um número positivo de apenas um dígito'
```

- Podem ser transformados em um único if

```
x = input('Digite um número: ')
if x > 0 and x < 10:
    print 'x é um número positivo de apenas um dígito'
```

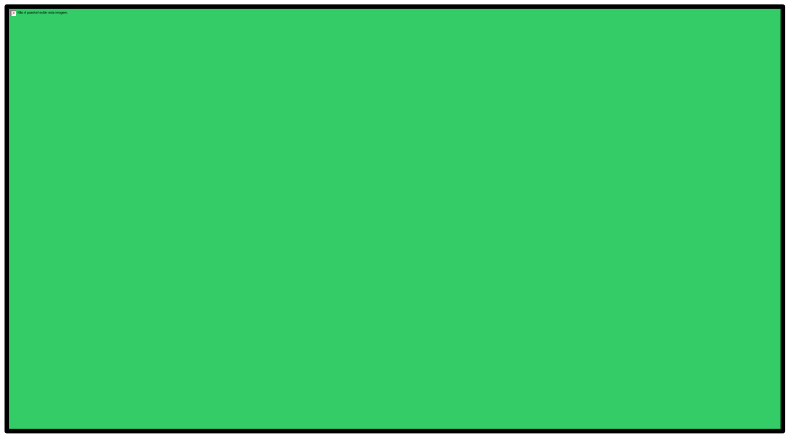
  - Operador lógico
    - **and**

Operadores Lógicos

- **and** → E
  - x > 0 **and** x < 10
    - Expressão somente será verdadeira se x for maior que zero **E** menor que 10
- **or** → Ou
  - x > 0 **or** x < -10
    - Expresão será verdadeira se x for maior que zero OU se x for menor que -10
- **not** → Negação
  - **not** (x > 0)

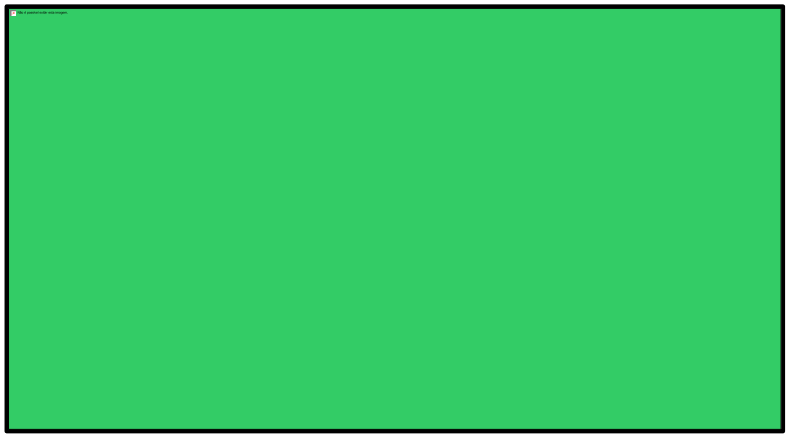
Se essa expressão for verdadeira, o **not** inverte o sentido da expressão e a torna falsa.

Tabela Verdade



- Uma expressão composta por **and** precisa que todas as subexpressões sejam verdadeiras para ser considerada verdadeira
  - x > 0 **and** x <= 0
    - **Nunca será!**
  - x < 100 **and** x > 0 **and** x%2 == 0

Tabela Verdade



- Uma expressão composta por **or** precisa que apenas uma das subexpressões seja verdadeira para ser considerada como tal
  - x > 0 **or** x <= 0
    - **Sempre será!**
  - x > 0 **or** x < 0

Funções Frutíferas

Valor de Retorno

- Recapitulando …
  - Algumas funções que já vimos produzem resultados

```
>>> p = math.pow(2, 3)
>>> x = p * math.sqrt(81)
```

  - Funções que retornam resultado geram um novo valor ao serem chamadas

```
import math
r = input('Digite o raio do círculo:')
area = 3.14 * math.pow(r, 2)
```

Funções Frutíferas

- E as funções que escrevemos?

- Qual a diferença entre imprimir o resultado e retorná-lo?

Funções Frutíferas

- Escreva uma função que receba dois números como entrada e imprima a soma dos dois.

```
def soma(x, y):
    print x + y

soma(2, 3)
```

```
def soma(x, y):
    return x + y


soma(2, 3)
z = soma(2, 8) * soma(2, 3)
```

- Funções que possuem retorno permitem que utilizemos o resultado de sua execução em outras expressões.

Funções Frutíferas

- A instrução **return** faz a execução retornar imediatamente da função e usar o valor em seguida como valor de retorno.

```
import math

def area(raio):
    return math.pi * raio ** 2
```


```
import math

def area(raio):
    a = math.pi * raio ** 2
    return a
```

Exercícios

- Faça uma função que receba três parâmetros e retorne a soma desses.

- Faça uma função que calcule e retorne a área de um retângulo.
  - **área = lado * altura**

- Faça uma função que calcule a área de um quadrado.
  - Faça e utilize nesta, uma função que calcule e retorne o quadrado de um número.

Funções Booleanas

```
def ehDivisivel(x, y):
    if x % y == 0:
        return True
    else:
        return False
```

```
def ehPositivo(x):
    if x > 0:
        return True
    else:
        return False
```

```
def ehDoPrimeiroQuadrande(x, y):
    if x > 0 and y > 0:
        return True
    else:
        return False
```

```
def estahEntre(x, y, x1, y1, x2, y2):
        if x >= x1 and x <= x2 and y >= y1 and y <= y2:
                return True
        else:
                return False
```