Laboratório de Programação 1

Aula 04

Mário Hozano
professor@hozano.com

Ciência da Computação
UFAL - Arapiraca

Relembrando a aula anterior...

- O que são valores, tipos e variáveis?
- Descreva os tipos *int, float e string*.
- Descreva os erros de sintaxe, de *runtime* e de semântica.
- O que é *debugging*?
- Para que serve as funções *print()*, *type()* e *input()*?
- Como realizar uma divisão inteira?
- O que são comentários em linguagem de programação?

Roteiro da aula

- Funções
- Argumentos
- Funções para Conversão de Tipos
- Funções Matemáticas
- Definindo Funções
- Fluxo de Execução
- Parâmetros e Variáveis Locais

Funções

- Nas aulas anteriores, aprendemos diversas funções

Função	Descrição
print()	Para exibir um texto na tela
type()	Para indicar o tipo de um valor
input()	Para o usuário inserir uma entrada manual a partir do teclado

Funções

- Nas aulas anteriores, aprendemos diversas funções

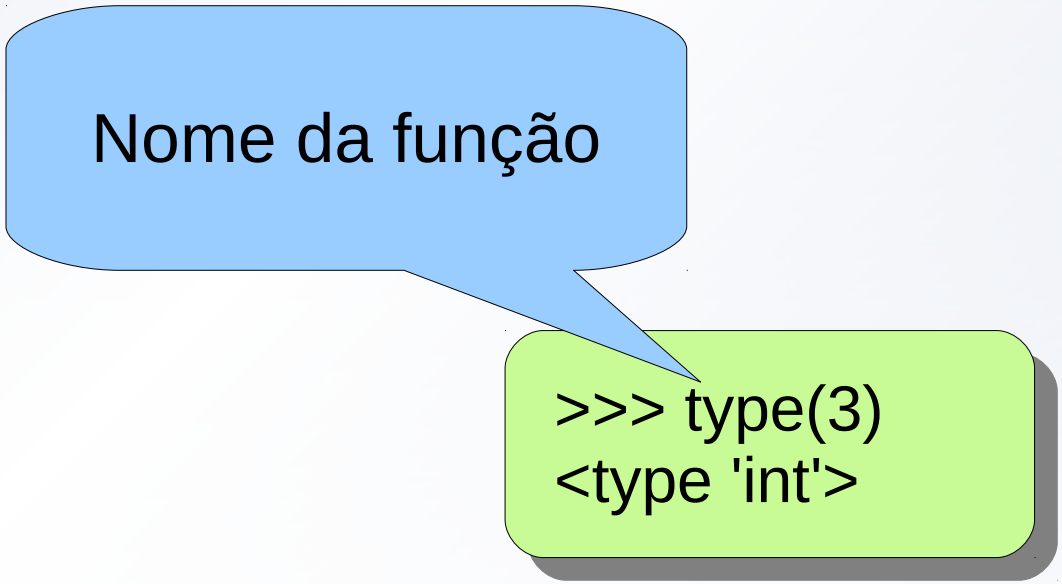
Função	Descrição
print()	Para exibir um texto na tela
type()	Para indicar o tipo de um valor
input()	Para o usuário inserir uma entrada manual a partir do teclado

Mas, o que é uma **função**?

Funções

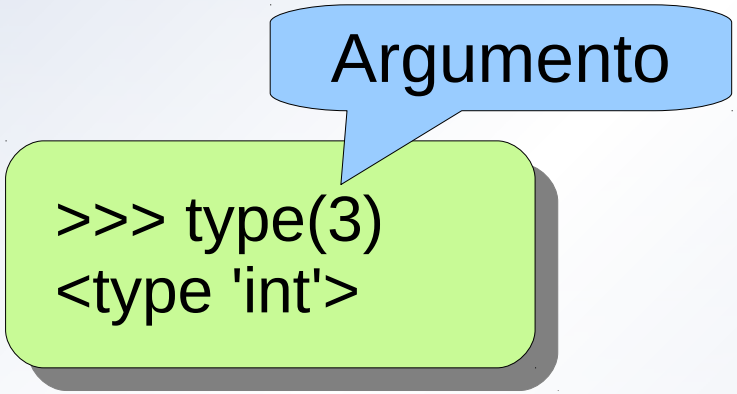
- Uma **função** é uma sequência de comandos que são referenciados por um nome

- Uma vez definida, uma função pode ser chamada em qualquer parte do programa



Funções - Argumentos

- A expressão entre os parênteses que sucedem o nome da função representa os **argumentos** (parâmetros?)



- Ao chamar uma função, deve-se indicar a quantidade de parâmetros de acordo como foi definido na função

```
>>> type(1,2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: type() takes 1 or 3 arguments
```

Funções - Retorno

- Uma função pode ou não retornar um valor após o seu processamento

- A função *print()* é um exemplo de função sem retorno

```
>>> print(“UFAL”)
UFAL
```

- A função *type()* é um exemplo de função com retorno

```
>>>  a = type(“UFAL”)
>>> print(a)
<type 'str'>
```

Funções para Conversão de Tipos

- Existem funções para converter valores de um tipo para outro, caso seja possível

- A função *int()* retorna um valor inteiro a partir da conversão do argumento passado

```
>>> int(32.33)
32
>>> int(“32”)
32
>>> int(“UFAL”)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'UFAL'
```

Funções para Conversão de Tipos

- Existem funções para converter valores de um tipo para outro, caso seja possível
- A função *int* retorna um valor inteiro a partir da conversão do argumento passado

Conversão de float para int

```
>>> int(32.33)
32
>>> int(“32”)
32
>>> int(“UFAL”)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'UFAL'
```

Funções para Conversão de Tipos

- Existem funções para converter valores de um tipo para outro, caso seja possível
- A função *int* retorna um valor inteiro a partir da conversão do arg

Conversão de float para int

Conversão de string para int

```
>>> int(32.33)
32
>>> int(“32”)
32
>>> int(“UFAL”)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'UFAL'
```

Funções para Conversão de Tipos

- Existem funções para converter valores de um tipo para outro, caso seja possível
- A função *int* retorna um valor inteiro a partir da conversão do arg

Conversão de float para int

Conversão de string para int

```
>>> int(32.33)
32
>>> int(“32”)
32
>>> int(“UFAL”)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'UFAL'
```

Impossível conveter a string “UFAL” para int

Funções para Conversão de Tipos

- As funções *float()* e *str()* realizam trabalho análogo ao da função *int()*, transformando valores para *float* e *string* respectivamente

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

Funções Matemáticas

- Python possui **módulos** que concentram diversas funções e variáveis específicas
- Um destes módulos (*math*) reúne diversas funções e varíaveis específicas para cálculos matemáticos
- Para o usar o módulo *math* deve-se importá-lo com o comando *import math*

```
>>> import math
```

- Este comando cria no programa um objeto de módulo que pode ser referenciado pelo nome *math*

Funções Matemáticas

```
>>> import math                 # Importando o módulo
>>> math.pi                     # O valor de PI
3.141592653589793
>>> math.sin(math.pi / 2)       # Calcula o seno
1.0
```

Funções Matemáticas

```
>>> import math                # Importando o módulo
>>> math.pi                    # O valor de PI
3.141592653589793
>>> math.sin(math.pi / 2)      # Calcula o seno
1.0
```

Para acessar uma função ou variável de um objeto, utiliza-se um ponto

Funções Matemáticas

```
>>> import math                 # Importando o módulo
>>> math.pi                     # O valor de PI
3.141592653589793
>>> math.sin(math.pi / 2)       # Calcula o seno
1.0
```

Para acessar uma função ou variável de um objeto, utiliza-se um ponto

É possível também importar apenas um subconjunto de funções e variáveis de um objeto
Ex.: from math import pi

Funções Matemáticas

Como saber todas as funções
e variáveis de um módulo ?

Funções Matemáticas

- A função *dir()* retorna todos os objetos (funções e variáveis) de um objeto passado como argumento

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

Como saber se o objeto *math.log* é uma função, ou uma variável ?

Funções Matemáticas

- Uma outra forma de visualizar as funções e variáveis de um objeto é utilizando a função *help()*

- No modo *shell*, o interpretador irá conduzir o programador a um ambiente de texto que indica funções e variáveis de um objeto

Após importar o módulo math no *shell*,
execute o comando *help(math)*
(para sair do help, digite **q**)

Definindo Funções

- Python oferece várias funções para usos genéricos
- Desenvolvedores precisam criar funções específicas para sua aplicação
- Em Python, a definição de novas funções é promovida com a palavra-chave **def**
- Uma função tem que ser definida antes de ser chamada. Caso contrário, o interpretador irá apresentar um erro de nome não definido.

Definindo Funções

- Programa com uma função simples

funcao.py

```
def bem_vindo():
    print(“Seja bem Vindo!”)
    print(“Tchau!”)

bem_vindo()
```

Definindo Funções

- Programa com uma função simples

Nome da função

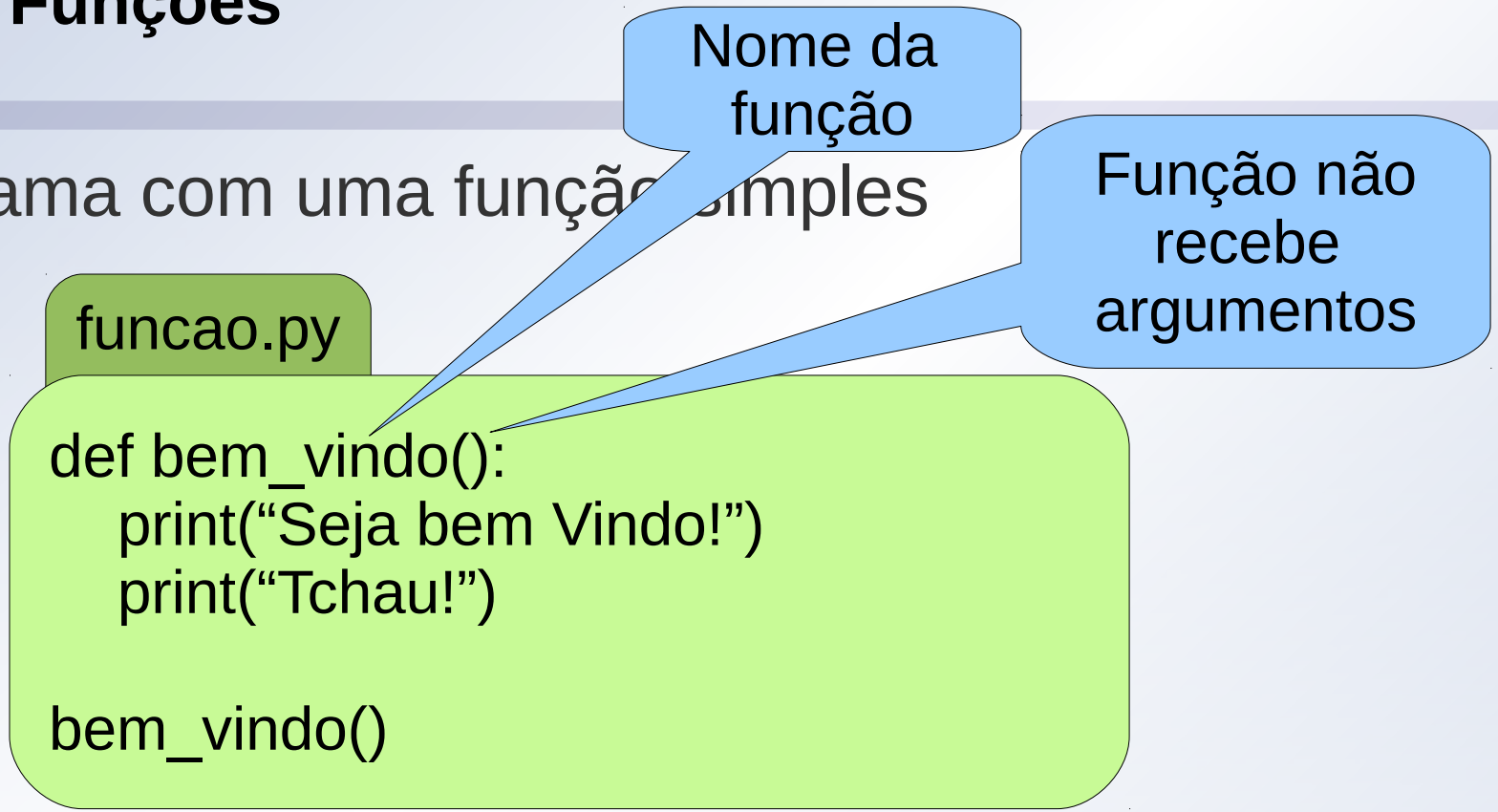
funcao.py

```
def bem_vindo():
    print(“Seja bem Vindo!”)
    print(“Tchau!”)

bem_vindo()
```

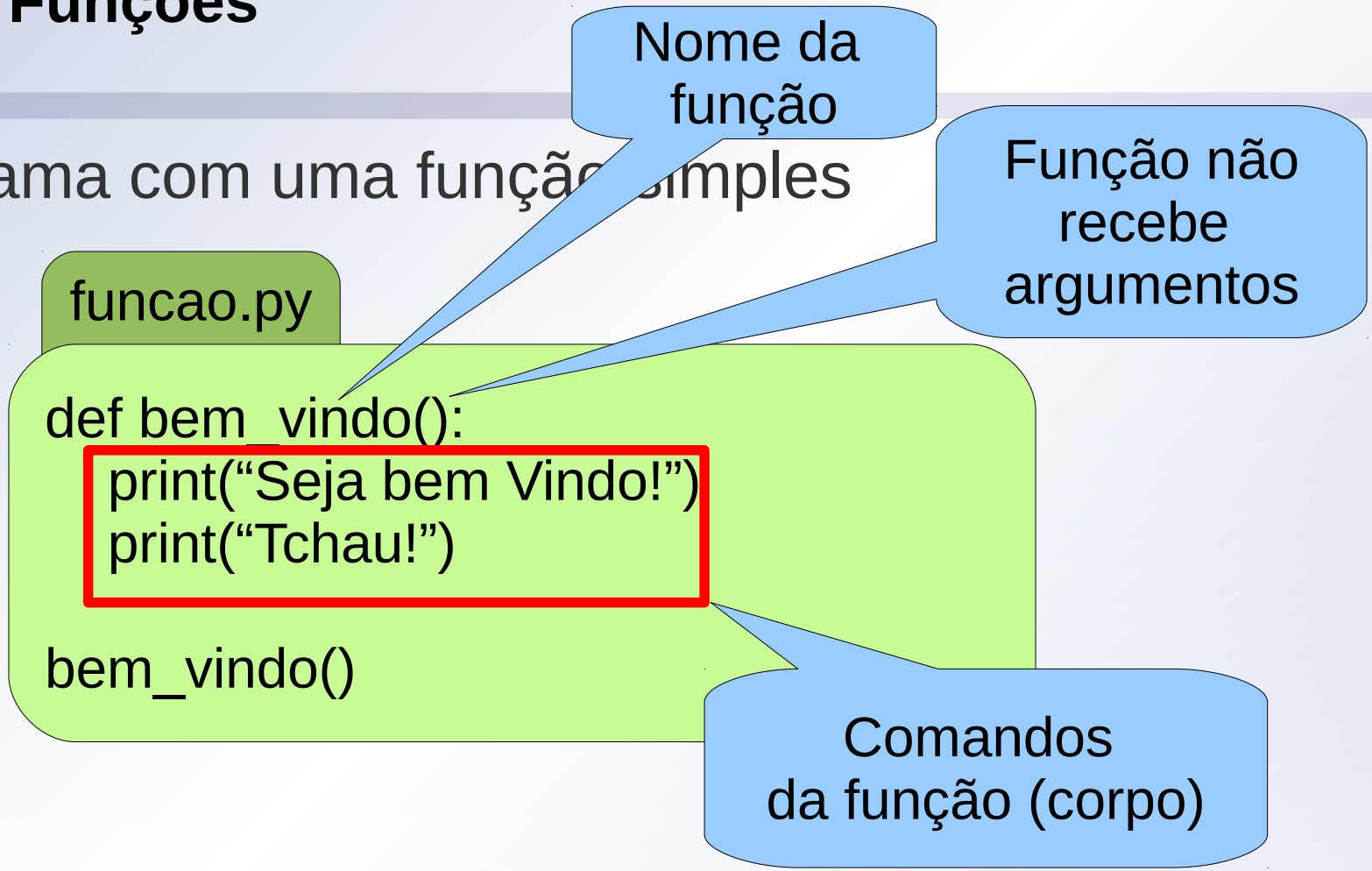
Definindo Funções

- Programa com uma função simples



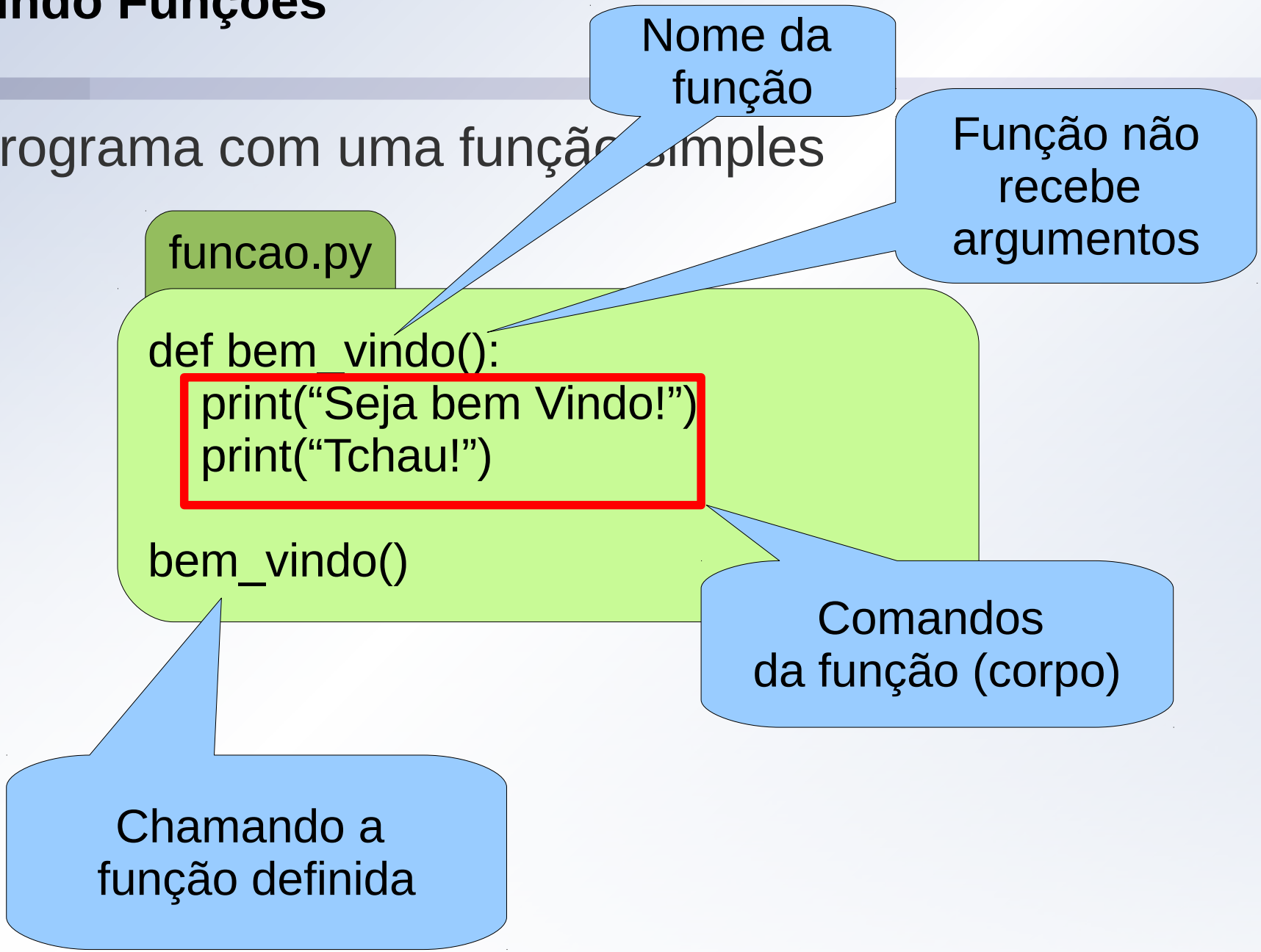
Definindo Funções

- Programa com uma função simples



Definindo Funções

- Programa com uma função simples



Definindo Funções – Outro Exemplo

diferenca.py

```
def diferenca(num1, num2):
    return num1 - num2

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

print(“A diferença dos números: “, diferenca(a,b))
```

Definindo Funções – Outro Exemplo

diferenca.py

```
def diferenca(num1, num2):
    return num1 - num2

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

print(“A diferença dos números: “, diferenca(a,b))
```

Chamando a função com 2 argumentos

Definindo Funções – Outro Exemplo

As variáveis que recebem os argumentos são chamadas de **parâmetros**

diferenca.py

```
def diferenca(num1, num2):
    return num1 - num2

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

print(“A diferença dos números: “, diferenca(a,b))
```

Chamando a função com 2 argumentos

Definindo Funções – Outro Exemplo

As variáveis que recebem os argumentos são chamadas de **parâmetros**

diferenca.py

```
def diferenca(num1, num2):
    return num1 - num2

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

print(“A diferença dos números: “, diferenca(a,b))
```

Chamando a função com 2 argumentos

O resultado de uma função pode ser utilizado como argumento de outra (composição)

Definindo Funções – Outro Exemplo

As variáveis que recebem os argumentos são chamadas de **parâmetros**

diferenca.py

```
def diferenca(num1, num2):
    return num1 - num2

a = in“Digite um número: ”)
ite outro número: ”)

print(“A diferença dos números: “, diferenca(a,b))
```

RETURN ?

Chamando a função com 2 argumentos

O resultado de uma função pode ser utilizado como argumento de outra (composição)

Definindo Funções – Funções com Retorno

- Uma função pode ter um ou mais instruções de retornos, mas apenas 1 é utilizado por chamada
- O comando de retorno termina o fluxo de execução da função
- Os retornos das funções podem ser atribuídos a variáveis ou podem ser utilizados como argumentos de outras funções

```
def area_retangulo(base, altura):
    area = base * altura
    return area

print(area_retangulo(10, 5))
```

Definindo Funções – Fluxo de Execução

- A execução de um programa acontece de cima para baixo, linha por linha
- Definições de funções não alteram o fluxo de execução do programa.
- Os comandos das funções são executados apenas quando as funções são chamadas
- A chamada de uma função interrompe a execução de um programa para executar os comandos de uma função
- Após executar os comandos da função, o programa volta para o ponto em que a mesma foi chamada

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

resultado = soma(a, b)

print(“A soma dos números é: “, resultado)
```

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

resultado = soma(a, b)

print(“A soma dos números é: “, resultado)
```

O interpretador inicia a execução de cima para baixo

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

resultado = soma(a, b)

print(“A soma dos números é: “, resultado)
```

O interpretador inicia a execução de cima para baixo

O nome da função é registrado e o interpretador passa para o próximo comando depois da função

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

resultado = soma(a, b)

print(“A soma dos números é: “, resultado)
```

O interpretador aguarda a entrada do primeiro número

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

resultado = soma(a, b)

print(“A soma dos números é: “, resultado)
```

O interpretador aguarda a entrada do segundo número

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input("Digite um número: ")
b = input("Digite outro número: ")

resultado = soma(a, b)

print("A soma dos números é: ", resultado)
```

A função *soma* é chamada com os Argumentos *a* e *b*

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input("Digite um número
b = input("Digite outro núm

resultado = soma(a, b)

print("A soma dos números é: ", resultado)
```

Os valores de *a* e *b* são atribuídos aos parâmetros *num1* e *num2*

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

resultado = soma(a, b)

print(“A soma dos números é: “, resultado)
```

A soma é realizada e atribuída à variável *s*

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

resultado = soma(a, b)

print(“A soma dos números é: “, resultado)
```

O valor da variável *s* é retornado

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input("Digite um número: ")
b = input("Digite outro número: ")

resultado = soma(a, b)

print("A soma dos números é: ", resultado)
```

O valor retornado da função *soma* é atribuído à variável *resultado*

Definindo Funções – Fluxo de Execução

fluxo.py

```
def soma(num1, num2):
    s = num1 + num2
    return s

a = input(“Digite um número: ”)
b = input(“Digite outro número: ”)

resultado = soma(a, b)

print(“A soma dos números é: “, resultado)
```

O valor da variável *resultado* é impresso na tela junto com a mensagem dada

Parâmetros e Variáveis Locais

- Os parâmetros e variáveis criadas dentro das funções possuem escopo local
- Elas só “existem” quando o fluxo da execução está dentro da função
- Acessar as variáveis locais fora da função causa um erro no programa caso não exista uma outra variável com mesmo nome criada fora da mesma

Parâmetros e Variáveis Locais - Exemplo

local.py

```
def soma(num1, num2):
    a = num1 + num2
    c = 10
    return a

a = 3
b = 5

print(a)
resultado = soma(a, b)
print(a)
print(c)
```

O que será impresso aqui ?

E aqui ?

E aqui ?

Exercícios

1. Faça um programa que receba 2 números do usuário e realize operações matemáticas com funções definidas.

2. Faça uma função que receba a base e altura de um retangulo e retorne a área do mesmo

3. Faça uma função que receba o raio e retorne a área de um círculo

4. Faça uma função que receba o raio e retorne o volume de uma esfera