

Fundamentos de Programação

revisão e adendos

Tipos de dados

- ✓ Programas manipulam valores/dados (textos, números, ...) de tipos distintos.

int
2017

float
3.141592653589793

str
Mia#0

- ✓ O **tipo** de dado define o conjunto de valores válidos e disponibiliza um conjunto de operações que podem ser efetuadas.
- ✓ Em *Python* os dados são objetos de um determinado tipo.
- ✓ Todo objeto tem um **tipo** e um **valor**, que são determinados quando ele é criado e armazenado na memória no formato de seu tipo.
- ✓ É necessário conhecer como a linguagem Python lida com os tipos de dados para evitar erros como realizar operações incompatíveis com um determinado tipo.



Alguns tipos de dados nativos simples

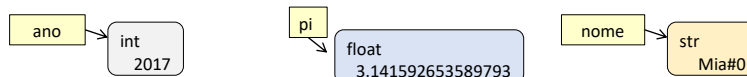
Dados	Tipo	Valores Válidos		Exemplos
Numéricos Inteiros	int	N Z	precisão fixa	5 101 6 -943 -4
Numéricos com parte fracionária	float	Q R	Precisão variável! mantissa (m) e expoente (e)	15.3 -0.37 6. .37 15e-5
Textuais ou Alfanuméricos	string	Sequência de letras, dígitos, símbolos de pontuação e especiais	igual ao número de caracteres da sequência	'a' '?' '6' '101' '12@34#\$' 'Informática Puc-Rio' "banana"

3



Variável

- ✓ Uma variável é uma associação entre um nome e um valor.



- ✓ Pode ser utilizada e alterada pelo programa, devendo ser criada antes de ser usada.
- ✓ O nome de uma variável pode ter um ou mais caracteres. Regras:
 - 1º caractere: letra ou *underscore*('_')
 - Demais caracteres: letras, números ou *underscore*('_')
 - Letras maiúsculas e minúsculas são consideradas diferentes;
 - **Proibido**: caracteres especiais, espaços ou palavras reservadas
- ✓ Em Python, o tipo da variável é o tipo do valor associado → operações mudam de acordo com o tipo.

4

Atribuição (1/5)

Associa o nome de uma variável ao valor resultante da avaliação de uma expressão.

`nomeVariável = expressão`

- ✓ Na 1ª atribuição: a variável é criada e seu conteúdo referencia o valor resultante da expressão.
- ✓ Nas demais atribuições: o valor da variável é substituído pelo resultado da expressão.
- ✓ Sempre que o nome de uma variável for utilizado em uma expressão ele será automaticamente substituído pelo seu valor.

5

Atribuição (2/5)

`nomeVariável = expressão`

→ Valor constante
→ Variável ou retorno da chamada de uma função
→ Expressão : <operando> operador <operando>

expressão aritmética: <operando> operador aritmético <operando>

Operação	Operador Aritmético
Soma	+
Subtração	-
Multiplicação	*
Divisão	/
Divisão de inteiros	//
Resto da divisão	%
Potenciação	**
Negativo	-

6

Expressão aritmética

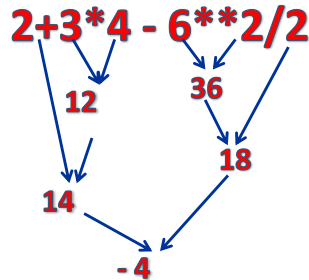
<operando> operador aritmético <operando>

A expressão é avaliada de acordo com a prioridade dos operadores



- 1) Exponenciação (**)
- 2) Multiplicação e Divisão (*, /, //, %)
- 3) Soma e Subtração (+, -)

expressão aritmética:



A ordem de avaliação pode ser modificada por parênteses

7

Atribuição (3/5)

Operações válidas com total → operações do tipo inteiro (+, -, *, /, //, %, **)

```
import math
```

```
total = 100
```

```
peso = 10.5
```

```
angulo = math.pi
```

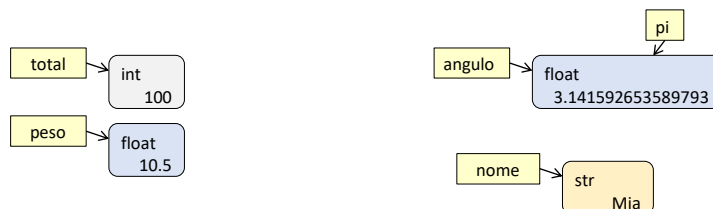
```
nome = 'Mia'
```

Cria variável total que referencia o objeto inteiro 100

Cria variável peso que referencia o objeto real 10.5

Cria variável angulo que referencia o objeto real pi

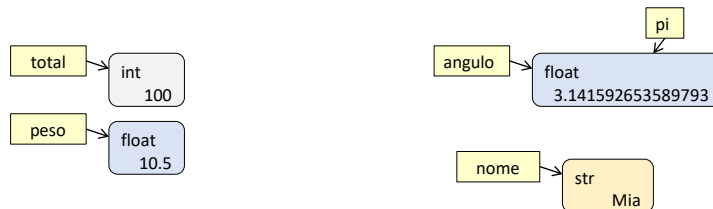
Cria variável nome que referencia o objeto string "Mia"



8

Atribuição (4/5)

```
import math
total = 100          # Cria variável total que referencia o objeto inteiro 100
peso = 10.5          # Cria variável peso que referencia o objeto real 10.5
angulo = math.pi     # Cria variável angulo que referencia o objeto real pi
nome = 'Mia'          # Cria variável nome que referencia o objeto string "Mia"
total = 'Não sei...'
```

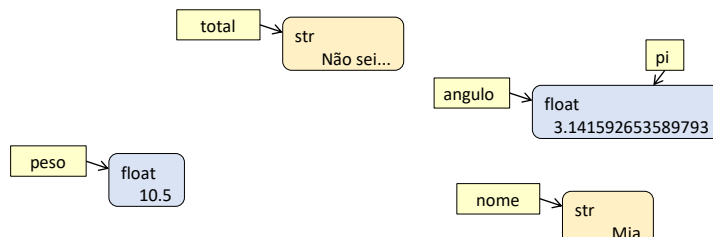


9

Atribuição (5/5)

Operações válidas
com total →
operações do tipo
string
(+, *)

```
import math
total = 100          # Cria variável total que referencia o objeto inteiro 100
peso = 10.5          # Cria variável peso que referencia o objeto real 10.5
angulo = math.pi     # Cria variável angulo que referencia o objeto real pi
nome = 'Mia'          # Cria variável nome que referencia o objeto string "Mia"
total = 'Não sei...'
```



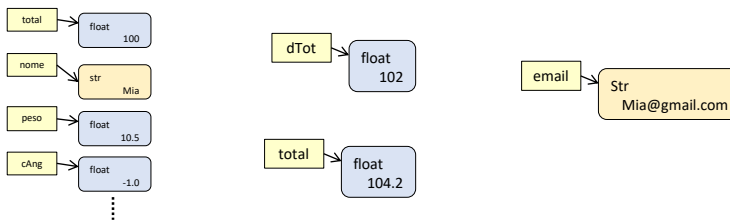
10

Operador +: adiciona ou concatena

- ✓ Se ambos os operandos são numéricos : soma os números
- ✓ Se ambos os operandos são textos : concatena os textos
- ✓ Demais casos: erro de execução (*TypeError: não converte tipos implicitamente*)

Exemplo:

```
dTot = total + 2          # dTot é criado e associado ao objeto int resultante de 100+2
total = total + 4.2      # total é associado ao objeto float resultante de 100+4.2
email = nome + '@gmail.com' # email é criado e associado ao objeto str da concatenação
```



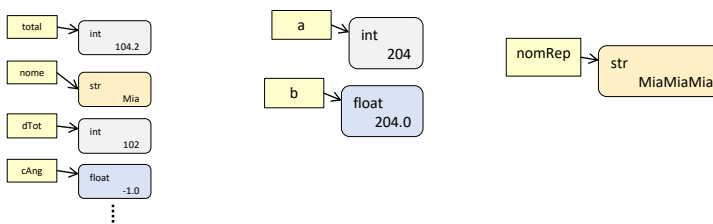
11

Operador *: multiplica ou replica

- ✓ Se ambos os operandos são numéricos: multiplica os números
- ✓ Se um operando é texto e outro nº inteiro: replica a string n° vezes
- ✓ Demais casos: erro de execução (*TypeError: não converte tipos implicitamente*)

Exemplos:

```
a = dTot * 2          # dTot é criado e associado ao objeto int resultante de 100+2
b = dTot * 2.0        # total é associado ao objeto float resultante de 100+4.2
nomRep = nome * 3      # nomRep é criado e associado ao objeto str com 'Mia' replicado 3 vezes
```



12

Mais sobre strings: índices

- ✓ Strings são sequências de caracteres.
- ✓ As posições dos caracteres na sequência são numeradas por índices:
 - ✓ da esquerda para direita, iniciando em 0.
 - ✓ da direita para a esquerda iniciando em -1.
- ✓ É possível acessar um caractere ou um intervalo (*fatia/slice*) da string, mas não modificá-los.

string s:									
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

13

Operações em strings: acesso aos elementos

- ✓ Um caractere é uma string de comprimento 1.
- ✓ Um elemento pode ser selecionado (indexado) pelo seu índice (sua posição na sequência) por meio do operador de indexação `[]`:

string s:

string s:									
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9
s[0]			s[3]						s[9]
↓			↓						↓
'U'			' '						'!'

14

Operações em strings: fatias

string[a:b:n]

Seleciona um intervalo (fatia) da string da posição **a** (inclusive) até a posição **b** (exclusive) de **n** em **n**.

- Se **a** não for definido, será considerado como zero
- Se **b** não for definido, será considerado o tamanho da string.
- Se o intervalo **n** (entre os caracteres), não for definido, será 1.

string s:

-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

Exemplo:

s[0:3] → 'Uma'
s[5:] → 'rase!'
s[:5] → 'Uma f'

15

Exemplos de fatias de strings

string s:

-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

Qual a string resultante?

s[:5]	'Uma f'
s[4:]	'frase!'
s[-1:-3:-1]	'!e'
s[-1:-3]	''
s[-3:-1]	'se'
s[:]	'Uma frase!'
s[::2]	'Uafae'
s[0:4:2]	'Ua'
s[4:0:-2]	'fa'
s[4: :-2]	'faU'

16

Fatias (*slices*): resumo

Sintaxe

```
seq[a:b: n]
```

seq[a : b] - cria uma cópia de **a** (*inclusive*) até **b** (*exclusive*)

seq[a :] - cria uma cópia a partir de **a** (*inclusive*)

seq[: b] - cria uma cópia até **b** (*exclusive*)

seq[:] - cria uma cópia de todos os elementos

seq[a : b: n] - cria uma cópia de **a** (*inclusive*) até **b** (*exclusive*) de **n** em **n** elementos

17

Variável string x Atribuição (1/6)

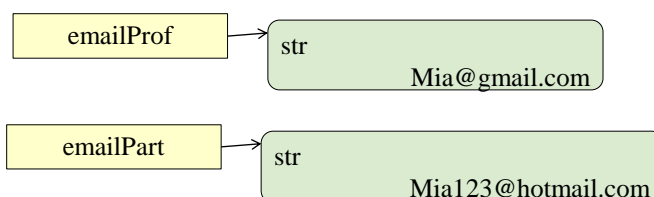
```
emailProf = 'Mia@gmail.com'           # Cria variável emailProf  
emailPart = 'Mia123@hotmail.com'      # Cria variável emailPart
```

18

Variável string: atribuição (2/6)

```
emailProf = 'Mia@gmail.com'           # Cria variável emailProf  
emailPart = 'Mia123@hotmail.com'      # Cria variável emailPart
```

Representação

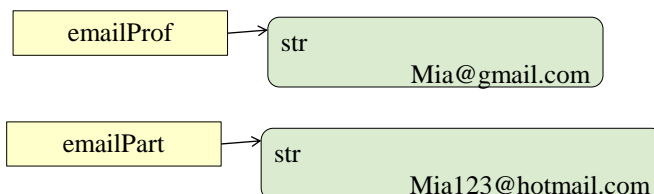


19

Variável string: atribuição (3/6)

```
emailProf = 'Mia@gmail.com'           # Cria variável emailProf  
emailPart = 'Mia123@hotmail.com'      # Cria variável emailPart  
emailProf = 'Mia@puc-rio.br'          # Associa outro valor à variável emailProf
```

Representação

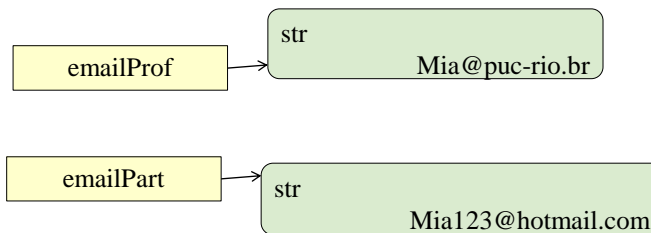


20

Variável string: atribuição (4/6)

```
emailProf = 'Mia@gmail.com'           # Cria var emailProf
emailPart = 'Mia123@hotmail.com'      # Cria var emailPart
emailProf = 'Mia@puc-rio.br'          # Associa outro valor à variável emailProf
```

Representação

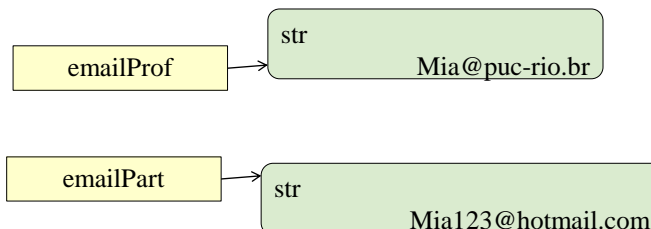


21

Variável string: atribuição (5/6)

```
emailProf = 'Mia@gmail.com'           # Cria var emailProf
emailPart = 'Mia123@hotmail.com'      # Cria var emailPart
emailProf = 'Mia@puc-rio.br'          # Associa outro valor à variável emailProf
emailProf[0] = 'a'
```

Representação



22

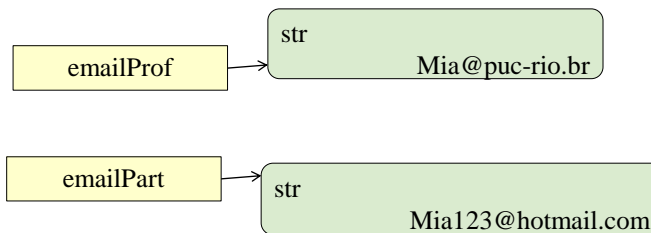
Variável string: atribuição (6/6)

```
emailProf = 'Mia@gmail.com'           # Cria var emailProf
emailPart = 'Mia123@hotmail.com'      # Cria var emailPart
emailProf = 'Mia@puc-rio.br'          # Associa outro valor à variável emailProf
emailProf[0] = 'a'                     #ERRO!!!!
```

emailProf[0]='a'

TypeError: 'str' object does not support item assignment

Representação



23

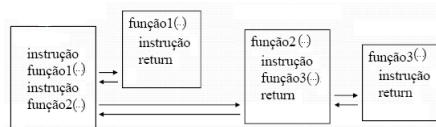
Funções

Uma **função** é uma sequência de instruções (bloco de código) independente, que realiza uma tarefa específica.

- ✓ Em geral computa valores a partir de valores recebidos.

Uma função pode receber e/ou retornar valores.

- ✓ As funções são "invocadas" (chamadas/ativadas) pelo nome, a partir de outra função, módulo ou diretamente no interpretador.
- ✓ Quando uma função termina sua execução, o controle retorna para o ponto de onde ela foi chamada (invocada/ativada).



24

Motivando o uso de funções

Desenvolvimento da solução por partes

- ✓ Permite pensar no problema em diversos níveis
- ✓ Modularização
- ✓ Testes independentes – facilita a correção de erros

Reuso

Mais legível (código menor)

Facilita a manutenção

25

Definindo funções (1/6)

Definição da função inicia com "def"

```
def funcao_qualquer(v1, v2, ... v2):  
    """ texto de documentação """  
    comando  
    ...  
    comando  
    return algo
```

o cabeçalho
termina com :

26

Definindo funções (2/6)

Definição da função inicia com "def"

Nome da função

```
def funcao_qualquer(v1, v2, ... v2):  
    """ texto de documentação """  
    comando  
    ...  
    comando  
    return algo
```

o cabeçalho
termina com :

27

Definindo funções (3/6)

Definição da função inicia com "def"

Nome da função

Parâmetros

```
def funcao_qualquer(v1, v2, ... v2):  
    """ texto de documentação """  
    comando  
    ...  
    comando  
    return algo
```

o cabeçalho
termina com :

28



Definindo funções (4/6)

Definição da função inicia com "def"

Nome da função

Parâmetros

Identação

```
def funcao_qualquer(v1, v2, ... v2):  
    """ texto de documentação """  
    comando  
    ...  
    comando  
    return algo
```

o cabeçalho termina com :

29



Definindo funções (5/6)

Definição da função inicia com "def"

Nome da função

Parâmetros

Identação

```
def funcao_qualquer(v1, v2, ... v2):  
    """ texto de documentação """  
    comando  
    ...  
    comando  
    return algo
```

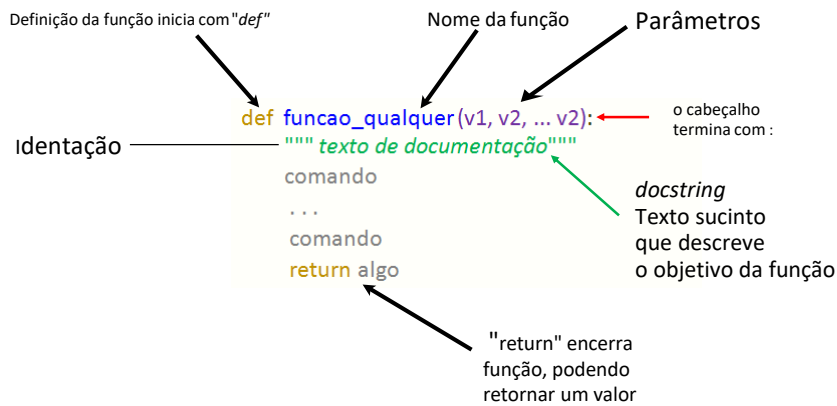
o cabeçalho termina com :

"return" encerra a função, podendo retornar um valor

30



Definindo funções (6/6)



31



Funções: sintaxe

Formato geral:

```
def funcao_qualquer (v1, v2, ... v2):  
    """ texto de documentação """  
    comando  
    comando  
    return algo
```

Em que:

- **nome** : nome associado à sequência de instruções.
- **v1..vn**: *parâmetros* - nomes associados aos valores transmitidos à função na sua chamada e necessários para realizar sua tarefa
 - Uma função pode ter 0, 1 ou mais parâmetros.
- **comandos**: sequência de instruções que será executada quando a função é invocada.

O comando **return <exp>**

- força o encerramento da função, retornando para quem a chamou.
- opcionalmente, devolve, um valor (**exp**) para a função de origem.

32

Observações sobre funções

- ✓ A definição da função deve ser feita antes de sua chamada, de modo que o interpretador Python reconheça o seu nome.
- ✓ Não pode haver funções e variáveis com o mesmo nome.
- ✓ Os valores recebidos são associados aos parâmetros na ordem em que foram definidos na chamada da função.
- ✓ A chamada da função deve incluir um valor para cada parâmetro (exceção: parâmetros com valores default)
- ✓ Uma função pode ter 0, 1 ou mais *returns*.
- ✓ Uma função pode chamar outra função.

```
def f(x,y):  
    return x-y  
def g(x,y):  
    return f(x+y,6)  
a=3  
b=4  
c=g(a,b)      #c:1
```

33

Observações

O que acontece???

```
def f(x,y):  
    return x-y
```

a=3

b=4

c=f(b,a) #c:1

d=f(a,b) #d:-1

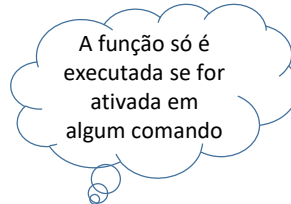
A declaração da função deve ser feita antes da sua chamada para que o Python reconheça o seu nome.

34

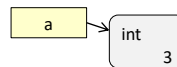
Funções: definição e uso (1/14)

O que acontece???

```
def f(x,y):  
    return x-y
```



```
→ a=3  
b=4  
c=f(b,a) #c:1  
d=f(a,b) #d:-1
```



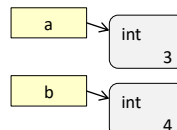
35

Funções: definição e uso (2/14)

O que acontece???

```
def f(x,y):  
    return x-y
```

```
a=3  
→ b=4  
c=f(b,a) #c:1  
d=f(a,b) #d:-1
```



36

Funções: definição e uso (3/14)

O que acontece???

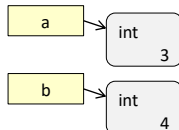
```
def f(x,y):  
    return x-y
```

a=3

b=4

```
→ c=f(b,a) #c:1  
d=f(a,b) #d:-1
```

A chamada da
função deve
incluir um valor
para cada
parâmetro



A função só é
executada se for
ativada em algum
comando

As funções são
ativadas
pelo nome

37

Funções: definição e uso (4/14)

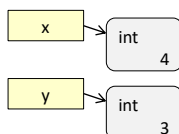
O que acontece???

```
→ def f(x,y):  
    return x-y
```

a=3

b=4

```
c=f(b,a) #c:1  
d=f(a,b) #d:-1
```

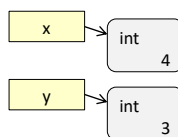


Os valores são recebidos
e associados aos
parâmetros na ordem
que estão na chamada.

38

O que acontece???

```
def f(x,y):  
    return x-y
```



```
a=3  
b=4  
c=f(b,a) #c:1  
d=f(a,b) #d:-1
```

39

O que acontece???

```
def f(x,y):  
    return x-y
```

```
a=3  
b=4  
c=f(b,a) #c:1  
d=f(a,b) #d:-1
```

1

40

Funções: definição e uso (7/14)

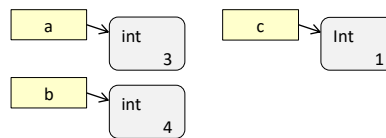
O que acontece???

```
def f(x,y):  
    return x-y
```

a=3

b=4

→ c=f(b,a) #c:1
d=f(a,b) #d:-1



41

Funções: definição e uso (8/14)

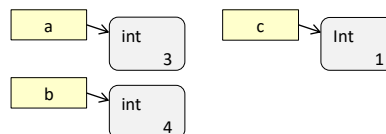
O que acontece???

```
def f(x,y):  
    return x-y
```

a=3

b=4

c=f(b,a) #c:1
→ d=f(a,b) #d:-1



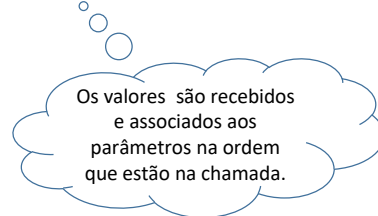
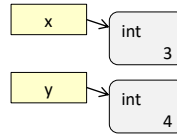
42

Funções: definição e uso (9/14)

O que acontece???

```
→ def f(x,y):  
    return x-y
```

```
a=3  
b=4  
c=f(b,a) #c:1  
d=f(a,b) #d:-1
```



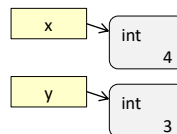
43

Funções: definição e uso (10/14)

O que acontece???

```
→ def f(x,y):  
    return x-y
```

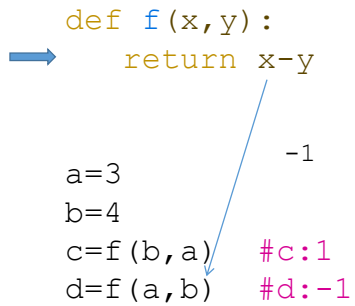
```
a=3  
b=4  
c=f(b,a) #c:1  
d=f(a,b) #d:-1
```



44

O que acontece???

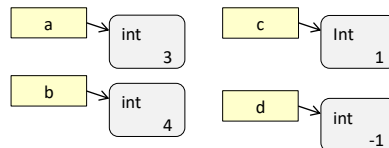
```
def f(x,y):  
    → return x-y  
  
a=3  
b=4  
c=f(b,a)    #c:1  
d=f(a,b)    #d:-1
```



45

O que acontece???

```
def f(x,y):  
    return x-y  
  
a=3  
b=4  
c=f(b,a)    #c:1  
→ d=f(a,b)  #d:-1
```



46

O que acontece???

```
def f(x,y):  
    return x-y  
  
a=3  
b=4  
c=f(b,a)    #c:1  
d=f(a,b)    #d:-1  
e=x-3
```

47

O que acontece???

```
def f(x,y):  
    return x-y  
  
a=3  
b=4  
c=f(b,a)    #c:1  
d=f(a,b)    #d:-1  
e=x-3
```

Os parâmetros e demais variáveis definidas dentro de uma função são **locais a ela**, isto é, só existem onde foram definidas.

As variáveis locais são descartadas quando a função é finalizada.

```
e=x-3  
NameError: name 'x' is not defined
```

48

Mãos na massa: string URL

Faça a função **nomeSite** que receba o código de uma disciplina da PUC, construa e retorne o endereço (URL) do site da disciplina.

Exemplo: código da disciplina: **inf1025**

String de retorno: www.inf.puc-rio.br/~inf1025

49

URL: desenvolvendo a solução (1/5)

Faça a função **nomeSite** que receba o código de uma disciplina da PUC, construa e retorne o endereço da site da disciplina.

Exemplo: código da disciplina: **inf1025**

String de retorno: www.inf.puc-rio.br/~inf1025

I. Qual o objetivo desta função?

Responder a quem a ativou o endereço do site da disciplina → retornar uma string.

50



URL: desenvolvendo a solução (2/5)

Faça a função **nomeSite** que receba o código de uma disciplina da PUC, construa e retorne o endereço da site da disciplina

Exemplo: código da disciplina: **inf1025**

String de retorno: www.inf.puc-rio.br/~inf1025

I. Qual o objetivo desta função?

Responder a quem a ativou o endereço do site da disciplina → retornar uma string.

II. Do que a função precisa para realizar sua tarefa?

51



URL: desenvolvendo a solução (3/5)

Faça a função **nomeSite** que receba o código de uma disciplina da PUC, construa e retorne o endereço da site da disciplina

Exemplo: código da disciplina: **inf1025**

String de retorno: www.inf.puc-rio.br/~inf1025

I. Qual o objetivo desta função?

Responder a quem a ativou o endereço do site da disciplina → retornar uma string

II. Do que a função precisa para realizar sua tarefa?

Do nome da disciplina (uma string).

52



URL: desenvolvendo a solução (4/5)

Faça a função **nomeSite** que receba o código de uma disciplina da PUC, construa e retorne o endereço da site da disciplina

Exemplo: código da disciplina: **inf1025**

String de retorno: www.inf.puc-rio.br/~inf1025

I. Qual o objetivo desta função?

Responder a quem a ativou o endereço do site da disciplina → retornar uma string

II. Do que a função precisa para realizar sua tarefa?

Do nome da disciplina (uma string)

III. Como ela realiza sua tarefa?

53



URL: desenvolvendo a solução (5/5)

Faça a função **nomeSite** que receba o código de uma disciplina da PUC, construa e retorne o endereço da site da disciplina

Exemplo: código da disciplina: **inf1025**

String de retorno: www.inf.puc-rio.br/~inf1025

I. Qual o objetivo desta função?

Responder a quem a ativou o endereço do site da disciplina → retornar uma string

II. Do que a função precisa para realizar sua tarefa?

Do nome da disciplina (uma string)

III. Como ela realiza sua tarefa?

1. Separa o código do departamento.
2. Monta o endereço do site usando o código.

54

URL: uma possível solução

```
def nomeSite(codDisc):  
    """ Nome do site dada disciplina"""  
    dep=codDisc[:3]  
    site='www.'+dep+'.puc-rio.br/~'+codDisc  
    return site
```

55

Escopo dos nomes (1/3)

```
import random  
def f(x,y):  
    a=random.randint(x,y)  
    return a+x-y
```

escopo 2 - Função

```
a=3  
b=4  
c=f(a,b)
```

escopo 1- Módulo

- ✓ O escopo define a visibilidade de nomes em blocos de códigos (módulo, função ou comando).

Espaço de nomes (name space) - nomes acessíveis em um ponto do programa.

- ✓ Um programa começa no escopo do módulo global (escopo1).
- ✓ Cada função/bloco acrescenta um escopo próprio – local (escopo 2).

56

Escopo dos nomes (2/3)

```
import random
def f(x,y):
    a=random.randint(x,y)
    return a+x-y
```

escopo 2 - Função

```
a=3
b=4
c=f(a,b)
```

escopo 1- Módulo

- ✓ Variável criada fora de um bloco é visível em qualquer lugar.
- ✓ Variável criada dentro de um bloco só existe se esse bloco for executado.
- ✓ Quando um nome é acessado, todos os escopos são consultados, do mais interno para o mais externo → variáveis globais podem ser ofuscadas por variáveis locais.

57

Escopo dos nomes (3/3)

```
import random
def f(x,y):
    a=random.randint(x,y)
    return a+x-y
```

escopo 2 - Função

Variáveis
Locais

a foi
'ofuscada',
pois foi
(re)definida
na função

Variáveis
Globais

```
a=3
b=4
c=f(a,b)
```

escopo 1- Módulo

Escopo 2 – Função f

Nomes:

Variáveis locais: a,x,y

Variáveis Globais: b,c

Instruções executadas só quando ativadas em um bloco de código.

Escopo 1 – Módulo

Nomes:

função f

Variáveis: a,b,c

Instruções imediatamente executadas pelo interpretador.

Não acesse variáveis globais a partir de uma função !!!!

58

Módulos

- ✓ Um módulo é, basicamente, um arquivo com a extensão `.py`, contendo funções, variáveis e constantes.
- ✓ Python fornece uma biblioteca padrão com inúmeros módulos.
- ✓ O programador pode criar seus próprios módulos, salvando seus arquivos com extensão `.py`.

Cuidado: para importar um módulo do programador, este deve estar salvo na mesma pasta do módulo atual ou em um caminho conhecido do Python.

- ✓ Um módulo pode ser importado por outro módulo, por meio do comando `import`, para que este possa fazer uso da funcionalidades definidas no módulo importado.

59

Importando módulos

➤ `import móduloX`

- Insere uma referência para o `móduloX` no *namespace* do módulo atual.
- Todos os nomes definidos no `móduloX` podem ser acessados.
- `móduloX.função(...)` – ativa uma *função* do `móduloX`

➤ `from móduloX import *`

- Todas as definições do `móduloX` são inseridas diretamente no *namespace* atual.
- Basta referenciar o nome de uma função definida no `móduloX` para ativá-la.

➤ `from móduloX import f1, f2, f3`

- apenas as funções `f1`, `f2` e `f3` do `móduloX` são inseridas no *namespace* do módulo atual. Útil em módulos com muitas funções.
- Evita conflito de nomes com o *namespace atual* e reduz tempo de carga do módulo.

60