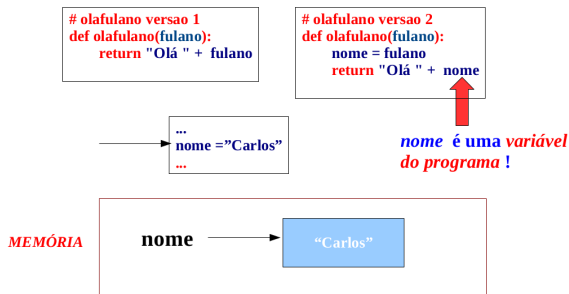


Computação 1 - Python

Aula 4 - Teórica: Variáveis e Atribuição, Strings

Variáveis e Atribuição

Variáveis são usadas para guardar dados intermediários nas funções.



Variáveis e Atribuição

- **Variável:** É uma maneira simbólica de fazer referência a dados armazenados na memória do computador.
- Toda variável engloba os seguintes aspectos, semelhantes aos parâmetros de uma função:

- **Nome (identificador):** é a representação simbólica da variável, que será utilizada pelo programador para fazer referência aos dados que ela armazena.

```
>>> x = 3
>>> x
3
```

- **Valor:** o que de fato está armazenado.
- **Tipo:** o tipo de dado que está armazenado.

Variáveis – Nomes de Variáveis

- Letras, números e underline (não começar por números)
 - `minhaVariavel = 1`
 - `minha_variavel = 1`
 - `minhaVariavel2 = 2`
 - `minha_variavel_2 = 2`
- **Dica:** em programas muito grandes e complexos, escolha (se possível) nomes que descrevam o significado da variável. Exceto em programas muito simples ou exemplos didáticos, evite nomes genéricos como “x”, “y”, “a”, etc.

Variáveis e Atribuição

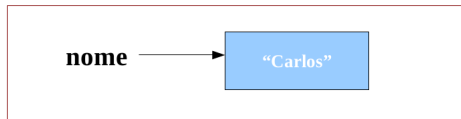
- **Atribuição:** O símbolo `=` é usado para atribuir um valor a uma variável.

var = valor

var1, var2, ..., varN = valor1, valor2, ..., valorN

```
...  
nome = "Carlos"  
return "Olá " + nome
```

MEMÓRIA



Atribuindo valores a variáveis

No interpretador python:

```
>>>a=1 # atribuo o valor 1 a variável a
>>> a # dá o valor armazenado em a
1
>>>a=2*a # armazeno na variável a o valor que está em a multiplicado por 2
>>>a # dá o valor armazenado em a
2
```

Atribuição Múltipla

```
>>> a,b,c = 1,2,3
```

```
>>> a
```

```
1
```

```
>>> b
```

```
2
```

```
>>> c
```

```
3
```

Como criar e usar uma variável ?

Uma variável é criada com um comando de atribuição:

$$\textit{variavel} = \textit{valor}$$

Um **alias** é um identificador que se refere a uma variável existente.

```
>>> x = 4
```

```
>>> y = x
```

A variável *y* é um **alias** para a variável *x*. Portanto, *y* possui o mesmo valor e aponta para o mesmo endereço de *x*.

Variáveis – Alias

O que acontece se atribuirmos um novo valor a “x”?

```
>>> x = 4
>>> y = x
>>> x = 5
>>> y
4
```

y permaneceu inalterada!!

- O que aconteceu foi algo bastante sutil (e bizarro): x é do tipo *int*, que é um tipo imutável (falaremos sobre isso mais tarde).
- Ao escrevermos “ $x = 5$ ”, em vez de modificar a variável x já existente, simplesmente criamos outra variável com o nome x e atribuímos a ela o valor 5. A variável x antiga é jogada fora.
- Como y era um alias para a variável x antiga, seu valor permaneceu inalterado.

Variáveis e Atribuição

Qual a diferença entre as funções abaixo ?

```
def testea( ):  
    a = 10  
    a,b=3*a,a  
    return a,b
```

```
def testea2( ):  
    a=10  
    a=3*a  
    b = a  
    return a,b
```

ATENÇÃO !

Variáveis – Tipo

- Python é uma linguagem dinamicamente tipada ou fracamente tipada.
- O tipo é atribuído de acordo com o valor atribuído à variável. Não é necessário *declarar previamente* o tipo.

```
>>> x = 4
>>> type(x)
<type 'int'>
```

- O tipo de uma variável pode mudar depois de alguma operação ou nova atribuição.

```
>>> x = complex(x)
>>> type(x)
<type 'complex'>
```

Variáveis – Escopo

- **Escopo:** onde a variável existe e onde ela deixa de existir.
- As variáveis definidas dentro de uma função são ditas **variáveis locais**, porque não podem ser acessadas fora da função.

```
def produtoSomaDiferenca(a,b):  
    x = a + b  
    y = a - b  
    return x*y
```

- As variáveis x e y são locais, pois só existem dentro da função. Depois que a função é executada, elas são destruídas.
- Dizemos que a função é o escopo de x e y .
- Tentar chamá-las fora da função ocasionaria um erro.

Exercícios

1. Faça uma função que receba dois parâmetros, sendo eles a hora e o minuto corrente, e informe quanto tempo (em minutos) se passou desde o início do dia. Faça o chinês para os seguintes valores:

Entrada	Valor de Retorno
(3,25)	?
(?,?)	882

2. Faça uma função que receba o valor inicial e a razão e imprima os 4 primeiros elementos da progressão geométrica correspondente. Experimente resolver o problema com diferentes números de variáveis. Qual o mínimo?

Strings

- *Caracteres* são símbolos. Podem ser letras, números, caracteres especiais, e até o espaço em branco é um caractere.
Exemplo: 'a', '9', '#', ' '.

- Uma *string* é uma sequência de caracteres.

```
>>> a = 'abcd'  
>>> b = "1234"  
>>> c = "#$5a"  
>>> d = ''  
>>> e = ' '
```

- Comprimento de uma string: número de caracteres que ela contém.

```
>>> s = '123456'  
>>> len(s)  
6
```

Strings - Índices

- Todo caractere de uma string é **indexado**, começando do primeiro caractere (**índice 0**) à esquerda.
- **Notação:** string[indice]

Exemplo: var = "Pedro dos Santos"

Índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Var	P	e	d	r	o		d	o	s		S	a	n	t	o	s

↑ var[2] : 'd' ↑ var[9] : 's' ↑ var[15] : 's'

Strings - Fatiamento

- Separa trechos de uma string.
- **Notação:** `string[índice1:índice2]`
 - Retorna os caracteres desde o de `índice1` até o de `(índice2 - 1)`
 - Se `índice1` é omitido, é assumido 0.
 - Se `índice2` é omitido, é assumido o fim da string.

Strings - Fatiamento

Exemplo

```
>>> x = 'abcde'
```

```
>>> x[0:2]
```

```
>>> x [2:]
```

```
>>> x[:]
```

```
>>> x[-1:]
```

```
>>> x[:-1]
```

Strings - Fatiamento

Exemplo

```
>>> x = 'abcde'
>>> x[0:2]
'ab'
>>> x [2:]
'cde'
>>> x[:]
'abcde'
>>> x[-1:]
'e'
>>> x[:-1]
'abcd'
```

Strings - Fatiamento

Incremento: podemos usar incremento / decremento para selecionar os elementos de uma string.

`[start:end:step]`: vai do índice *start* até *end* (sem ultrapassá-lo, com passo *step*)

Exemplo

```
>>> x= "abcde"
```

```
>>> x[0:-1:2]
```

```
>>> x[3:0:-1]
```

Strings - Fatiamento

Incremento: podemos usar incremento / decremento para selecionar os elementos de uma string.

`[start:end:step]`: vai do índice *start* até *end* (sem ultrapassá-lo, com passo *step*)

Exemplo

```
>>> x= "abcde"  
>>> x[0:-1:2]  
'ac'  
>>> x[3:0:-1]  
'dcb'
```

Strings

- Elementos de uma string não aceitam o operador de atribuição.

```
>>> s = '123456'
```

```
>>> s[0] = '0'
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#1>", line 1, in <module>
```

```
    s[0]='0'
```

```
TypeError: 'str' object does not support item assignment
```

- Strings são, portanto, **imutáveis**. Ou seja, os dados contidos em uma string não podem ser alterados.

Strings - Recapitulando

- **Representação:** `s = "12346"` ou `s = '123456'`
- **len(s)** : retorna o tamanho de uma string.
- **Operador +:** concatena strings. Ex: `'ab' + 'cd' = 'abcd'`
- **Operador *:** repete strings. Ex: `'a'*5 = 'aaaaa'`
- **Fatias (Slices):** `[start:end:step]`

Exercício

1. Faça uma função que dado o nome de uma pessoa, retorne o número de letras do nome e a primeira letra do nome.
2. Faça uma função que dada uma palavra, retorna a palavra invertida.
3. Faça uma função que dada uma palavra, retorna os caracteres nas posições ímpares.
4. Faça uma função que recebe duas strings e retorna a concatenação delas, com exceção do primeiro caractere de cada uma. Exemplo: dadas as entradas 'abcd' e 'efghi', o valor de retorno será 'bcdfghi'.
5. Escreva uma função que receba uma string e retorne a concatenação de três cópias dos dois últimos caracteres. Exemplo, se a entrada for 'abcd', a saída deve ser 'cdcddcd'.
6. Faça uma função que recebe duas strings e retorna a concatenação delas, em ordem alfabética, com espaço no meio. Exemplo: dadas as entradas 'xbcd' e 'efghi', o valor de retorno será 'efghi xbcd'.

Computação 1 - Python

Aula 4 - Teórica: Variáveis e Atribuição, Strings