
# Treinamento

## Python/Zope/Plone

# Apresentação

Luiz Gustavo da Fonseca Ferreira

- Formação e experiência:
  - Ciência da Computação (UFMG)
  - Mestrado em Ciência da Informação (UFMG)
  - Doutorando em Ciência da Informação (UFMG)
  - Diretor de tecnologia da Paradigma
  - 6 anos de experiência com PZP

# Ementa

- Python
- Plone na visão do usuário
- Templates e Scripts Python no Plone
- Buildout
- Temas
- Viewlets e Portlets
- Tipos de conteúdo customizados
- Webservices
- Workflow
- SVN
- Manutenção
- Otimização
- Extras: Plone com banco de dados relacional (Relstorage) e Autenticando com LDAP




# Python

## Iniciante

# I
# Introdução

## Módulo Python (Iniciante)

# O que é Python?

- Linguagem de programação criada em 1991 por Guido van Rossum.
- Projetada para ser simples e poderosa.
- Altamente legível.
- Interpretada e compilada.
- Orientada a objetos.
- Software livre.



# O que é Python?

- A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade.



# Quem usa Python?

- Atualmente Python é uma das linguagens mais utilizadas em grandes projetos no mundo.
- Engenheiros de software de empresas como Google afirmam que o uso de Python é essencial para o funcionamento de famosas ferramentas do Google.
  - “C++ quando necessitamos, Python quando podemos”.
- As ferramentas Google Drive, Picasa, Google+, Youtube, dentre outras da empresa americana Google utilizam a linguagem Python.

# Quem usa Python?

- Grande parte da API do Google é desenvolvida em Python.
- O Google App Engine (framework web online mantido pelo Google) oferece suporte total a Python e foi desenvolvido também em Python.
- Zope, um dos mais poderosos servidores de aplicação da atualidade foi desenvolvido em Python.

# Quem usa Python?

- Plone, que é uma aplicação do Zope, foi desenvolvido em Python e é utilizado largamente em diversos web sites: Governo Brasileiro, Portal da Câmara dos Deputados, PyCon Brasil, Novell, Free Software Foundation, etc.
- Diversas empresas que fabricam distribuições Linux utilizam softwares personalizados feitos em Python: Novell (SuSE e Open SuSE), Canonical (Ubuntu), Red Hat, dentre outras.
- A Microsoft também investe fortemente em Python, no desenvolvimento da linguagem para o .NET Framework

# Quem usa Python?

- Algumas empresas que patrocinam a *Python Software Foundation* (PSF):
  - Canonical
  - Google
  - Microsoft
  - O'Reilly
  - Sun Microsystems
  - Zope Corporation



# Por que usar Python?

- Python é portável, você não precisa compilar seu programa cada vez que muda de sistema operacional, basta apenas que exista um interpretador Python instalado.
- Python é open-source com licença flexível, há uma comunidade que desenvolve a linguagem, que está sempre aprimorando-a e qualquer um de nós pode fazer parte dessa comunidade. A licença flexível nos permite desenvolver softwares proprietários com ferramentas livres.



# Por que usar Python?

- Python é simples e auto-descritiva, algumas linguagens de programação têm sintaxe complexa ou difícil de memorizar.
- Vejamos alguns exemplos de código para declarar uma classe (Faladora) com as seguintes características:
  - tem um método público dizerOi
  - recebe uma string que representa o nome de alguma pessoa e
  - escreve uma mensagem de boas vindas para esta pessoa na tela. Instanciaremos um objeto falador da classe e chamaremos este método.



# PHP

```php
<?php
   class Faladora {
      public function dizerOi($nome){
         echo "Olá $nome! <br />";
      }
   }

   $falador = new Faladora();
   $falador->dizerOi(“Francisco”);
?>
```

# C++

```
class Faladora {
  public:
    void dizerOi(string* nome);
};

void Faladora::dizerOi(string* nome){
    cout << “Olá” << nome->c_str() << “!” <<
endl;
}
int main(void) {
  Faladora* falador = new Faladora();
  falador->dizerOi(new string(“Francisco”));
}
```

# Java

```
package minicurso;

public class Faladora {
   public void dizerOi(String nome) {
      System.out.println("Olá “ + nome +
"mundo!");
   }

   public static void main(String[] args) {
      Faladora falador = new Faladora();
      falador.dizerOi(“Francisco”);
   }
}
```



# Python

```
class Faladora:
    def dizerOi(self, nome):
        print("Olá %s!") %(nome)

falador = Faladora()
falador.dizerOi("Francisco")
```



# Por que usar Python?

- Note que é simples e objetivo! :)
- A presença de um interpretador interativo (assim como Ruby) também é uma vantagem, pois é interessante para iniciantes utilizarem um “interpretador de comandos”.



# Utilizando o interpretador

- Todas as distribuições Linux que trazem interface gráfica hoje vêm com o Python instalado, já que diversos softwares para este sistema operacional são feitos em Python. Desta forma, geralmente não é necessário instalar o Python em distribuições Linux.



# Utilizando o interpretador

- Para executar o interpretador no Linux:
  - Abra o terminal
  - Digite o comando python

# Utilizando o interpretador

- Comandos básicos
  - help()
  - import this
  - print “Olá mundo!”



# Exemplos

- Digite os seguintes exemplos no interpretador:
  - 10+5
  - 10-5
  - 3*5
  - 12**2
  - 100/4
  - x = 40
  - x/2
  - (1+2)*3



# Exercícios

No interpretador:

- Some 12, 20 e 40.
- Subtraia 50 de 58.
- Multiplique 10 por 5.
- Divida 200 por 50.
- Calcule 5 elevado a 3 (Dica:  Usar **).



# II
# Tipos e Estruturas de Dados

## Módulo Python (Iniciante)

# Sintaxe

- Simples e objetiva! Não é preciso saber programação para entender este código, basta um conhecimento básico em inglês!

```
if x > y:

    grande = x
    pequeno = y

else:

    grande = y
    pequeno = x
```

# Strings

- É uma sequência de caracteres envolta por aspas simples ou duplas
- Exemplos:
  - “Hello world”
  - ‘Bom dia!’
  - “Uma string qualquer”
  - ‘a’

# Strings (Operações)

- Concatenação
  - “Hello” + “World” → “HelloWorld”
- Repetição
  - “UMBC” * 3 → “UMBCUMBCUMBC”
- Indexação
  - “UMBC”[0] → “U”
- Slicing (fatiamento)
  - “UMBC”[1:3] → “MB”
- Tamanho
  - len(“UMBC”) → 4
- Pesquisa
  - “e” in “Hello” → True



# Listas

- A list (lista) é uma das mais conhecidas estruturas de dados.
- Em Python, a lista tem estado e comportamento que faz uma espécie de mistura entre o comportamento de vetores (acesso rápido por índices) e listas (“iterabilidade”);

# Listas (Exemplo)

- alunos = []
- alunos.append("Monica")
- alunos.append("Cascao")
- alunos.append("Cebolinha")

- print alunos
- alunos.reverse()
- print alunos



# Listas (Métodos)

- Alguns métodos para manipular uma lista:
  - append(elemento): adiciona o elemento à lista
  - count(elemento): retorna a quantidade de elementos da lista
  - reverse(): inverte a ordem dos elementos na lista
  - remove(elemento): remove o elemento da lista



# Listas (Exemplos)

- teste = ["Nome1", "Nome2", "Nome3"]
- teste.append("Nome4")
- teste.append("Nome2")
- print(teste.count("Nome2"))
- teste.reverse()
- print(teste)



# Tuplas

- Tuplas são como as listas, a diferença é que tuplas são estruturas de dados imutáveis.
- Pode-se dizer que, em Python, uma string é uma tupla de caracteres.
- Vamos a um exemplo para melhor compreensão. Veremos primeiro o que não é possível fazer com tuplas.



# Tuplas

*TypeError: 'tuple' object does not support item assignment*

```
tupla = ("Teste 1", "Teste 2")
tupla[0] = "Teste 3"
tupla.append("Teste 4")
```

*AttributeError: 'tuple' object has no attribute 'append'*

# Tuplas

- Vale lembrar que da mesma forma que fazemos com strings, também podemos concatenar tuplas.

- tupla = ("Teste1", "Teste2", "Teste3")
- tupla += ("Teste4", "Teste5", "Teste6")

- tupla2 = ("Teste7", "Teste8", "Teste9")
- tuplas = tupla + tupla2



# Dicionários

- Um dicionário é uma estrutura de dados composta por um conjunto de duplas chave e valor.
- Têm este nome pois são praticamente a mesma coisa que um dicionário!
- Exemplo:
  - idades = {“João”: 32, “Carlos”: 22}



# Dicionários (Exemplo)

- idades = {‘Joao’: 20,  ‘Augusto’: 25}
- print(idades[‘Joao’])
- idades[‘Joao’] = 21
- print(idades[‘Joao’])
- idades[‘Maria’] = 35
- print(idades)

# Dicionários (Métodos)

- Alguns métodos de manipulação de dicionários:
  - get(indice): retorna o valor correspondente ao indice
  - values(): retorna uma lista com todos os valores do dicionário.
  - keys(): retorna uma lista com todas as chaves do dicionário.
  - clear(): remove todos os itens do dicionário.



# Dicionários (Exemplo)

- nascimento = {"Maria" : 1990, "José" : 1985}
- print(nascimento.get("Maria"))
- print(nascimento.values())
- print(nascimento.keys())
- nascimento.clear()
- print(nascimento)



# Variáveis

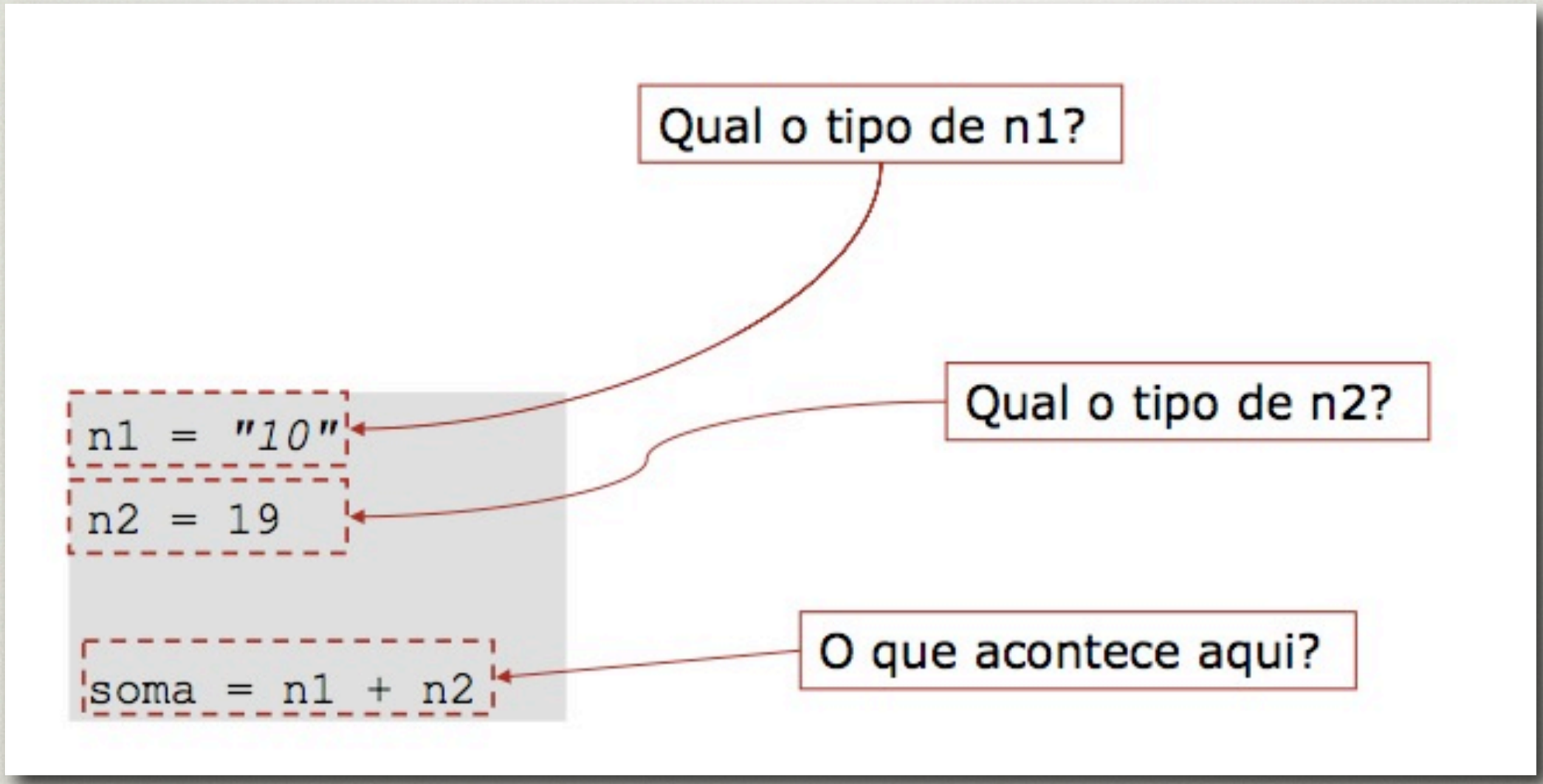
- O conceito de variável em Python é representado sempre por um objeto. Na verdade, Python é uma linguagem pura, ou seja, tudo em Python é um objeto.
- Toda variável é uma referência!
- Variáveis armazenam endereços de memória e não valores!

# Variáveis

- Python possui tipagem dinâmica e forte;
- Tipagem dinâmica significa que a variável assume o tipo de acordo com o valor atribuído;
- Tipagem forte significa que se uma variável é do tipo inteiro, ela deve ser tratada como um inteiro, ou convertida para ser tratada de outra forma;
- Para mudar o tipo de uma variável, devemos declará-la novamente;



# Exemplo



# Exercícios (String)

- Declare duas strings da seguinte forma:
- s1 = 'Bom '
- s2 = 'Dia'
- A) Concatenar as duas strings (Dica: '+')
- B) Retornar a letra 'm' de s1 através do seu índice (Dica: s1[x])
- C) Usar fatiamento para retornar o trecho 'om' de s1 (Dica: s1[x:y])
- D) Usar a função len(string) para retornar o tamanho das duas strings.
- E) Usar o operador in para testar se a letra 'm' existe em s1 (Dica: letra in string)



# Exercícios (Listas)

- Declare a seguinte lista:
- selecionados = [‘Maria’, ‘Renata’, ‘Augusto’]
- A) Adicionar ‘Fernando’ à lista (Dica: append)
- B) Usar a função len para retornar o tamanho da lista
- C) Retornar a string ‘Renata’ através do seu índice (Dica: lista[x])
- D) Remover ‘Augusto’ da lista (Dica: remove)
- E) Usar o operador in para testar se ‘Augusto’ e ‘Fernando’ existem em selecionados (Dica: string in lista)
- F) Inverta a ordem dos elementos na lista (Dica: reverse)

# Exercícios (Dicionários)

- Declare o seguinte dicionário:
- meses = {'Janeiro': 'January', 'Fevereiro': 'February'}
- A) Adicionar 'Março' = 'March' no dicionário (Dica: dic['chave']=valor)
- B) Usar o método keys para retornar as chaves do dicionário
- C) Usar o método values para retornar os valores do dicionário
- D) Retornar o valor da chave 'Janeiro' (Dica: dic['chave'])
- E) Remover a chave 'Janeiro' do dicionário Dica: del(dic['chave'])
- F) Usar o operador in para testar se 'Janeiro' e 'Fevereiro' existem em meses



# III
# Módulos e entrada de dados

## Módulo Python (Iniciante)

# Módulos

- Módulo é o agrupamento de comandos em um arquivo. Numa definição mais informal, módulo é um arquivo de código-fonte Python.
- Módulos são arquivos definidos com as extensões .py, .pyc e outras menos utilizadas.
- Nossos módulos terão sempre a extensão .py, pois se trata da extensão para código fonte.

# Módulos (Exemplo)

- Na pasta referente a este módulo, crie um arquivo chamado olamundo.py
- Usando o seu editor de texto, escreva o seguinte trecho no módulo:

```
# coding=utf-8

print('Olá mundo')
```

- Execute o módulo no terminal através do seguinte comando:

```
python olamundo.py
```

# Módulos (Exercício)

- Crie um módulo chamado fonte.py com o seguinte conteúdo:
- minha_variavel = 5

- Na mesma pasta, crie outro módulo chamado usuario.py com o seguinte conteúdo:

```
from fonte import minha_variavel

print(minha_variavel)
```

- Execute o módulo usuário (python usuario.py)

# Entrada de dados

- Agora vamos aprender como receber informações a partir do teclado
- Para isso, utiliza-se a função raw_input().
- Exemplo:

```
nome = raw_input('Digite o seu nome: ')
idade = raw_input('Digite a sua idade: ')
print('Nome: ' + nome)
print('Idade: ' + idade)
```

# Entrada (Exercício 1)

- Escreva um programa que solicite ao usuário o tamanho dos dois lados de um retângulo e calcule a área desse retângulo.
- A = lado1 * lado2
- Converter:
  - int(minha_string)



# IV

# Controle de fluxo

## Módulo Python (Iniciante)

# Desvio condicional (if)

- Desvio condicional é a forma de permitir a escolha de um entre múltiplos fluxos de execução por parte da máquina.
- O caso mais usado se dá através da instrução if.
- Veja um exemplo:

```
if temperatura < 20:
    print('Está fazendo frio!')
else:
    print('Não está fazendo frio!')
```

# Desvio condicional (Exemplo)

```
uma_lista = [1,3,5,7]

if 5 in uma_lista:
    print('o número está na lista')
else:
    print('o número não está na lista')
```

# Desvio condicional (Exercício 1)

Faça um programa que dada uma lista de tamanho 3, imprima o maior dos números na lista.

Exemplo, se a lista for:

uma_lista = [10, 15, 3]

Seu programa deve imprimir 15



# Desvio condicional (Exercício 2)

- Resolva o exercício anterior usando o operador and. Exemplo:

```
if <condição 1> and <condição 2>:
    <faça alguma coisa>
```



# Laços

- Também chamados de *loops*.
- São blocos de comandos executados repetidamente



# Laços Enquanto (while)

- São executados enquanto uma condição for verdadeira.
- Exemplo:
  - i = 0
  - while i<10:
  - print (‘i ainda é menor que 10’)
  - i += 1
  - print(‘i é igual a 10’)



# while (Exercício 1)

- Escreva um programa para imprimir os números entre 0 e 1000
- SAÍDA:
  - 0
  - 1
  - 2
  - ...
  - 998
  - 999
  - 1000

# while (Exercício 2)

- Escreva um loop while para gerar uma lista de 1s de tamanhos variados
- tamanho = 5
- lista = []
- <loop while a ser criado>
- print(lista)
- SAÍDA:
- [1, 1, 1, 1, 1]



# Laços Para (for)

- São executados para cada item de um determinado conjunto iterador (lista, tupla, dicionário, etc).
- Exemplo:
- umaLista = [1, 2, 'abc']
- for elemento in umaLista:
- print (elemento)



# for (Exercício 1)

- Escreva um programa para imprimir os números entre 100 e 200.
- Dica: usar a função range(num1, num2) - teste-a no interpretador interativo



# for (Exercício 2)

- Escreva um programa que some os números entre 1 e 10 usando um loop for (e imprima o total no final)



# Exercício extra

- Gere uma lista de números de tamanho N seguindo a sequência de Fibonacci.
- N será dado pelo usuário (usar input)
- Dica: lista[-1] se refere ao último item da lista e lista[-2] se refere ao penúltimo item da lista.
- Na sequência de Fibonacci, cada termo corresponde a soma dos dois anteriores. Exemplo:
- [1, 1, 2, 3, 5, 8, 13, ...]



# V

# Desafios

## Módulo Python (Iniciante)

# Metros x milímetros

- Escreva um programa que leia um valor em metros e o exiba convertido em milímetros
- Exemplo:
  - Entrada (em metros): 2
  - Saída: 2 metros equivalem a 2000 milímetros



# Aumento de salário

- Faça um programa que calcule o aumento de um salário. Ele deve solicitar o valor do salário e a porcentagem do aumento. Exiba o valor do aumento e do novo salário.
- Exemplo:
  - Entrada:
    - Salário (R$): 2000
    - Aumento (%): 20
  - Saída:
    - Aumento (R$): 400
    - Novo salário (R$) : 2400



# Life, the Universe and Everything

- Faça um programa que acumule os números da entrada em uma lista e pare de acumular após ler o número 42. Imprima a lista no final.
- Exemplo:
  - Entrada:
    - Digite um número: 20
    - Digite um número: 103
    - Digite um número: 8
    - Digite um número: 42
  - Saída:
    - [20, 103, 8, 42]

# Converter para segundos

- Escreva um programa que leia a quantidade de dias, horas, minutos e segundos do usuário. Calcule o total em segundos.
- Exemplo:
  - Entrada
    - Dia(s): 1
    - Hora(s): 3
    - Minuto(s): 1
    - Segundo(s): 0
  - Saída:
    - Segundo(s): 97260

# Números primos

- Faça um programa para calcular se um número é primo
- Primo: número divisível apenas por 1 e ele mesmo.
- Lógica: calcular o resto da divisão do número por todos os números menores que ele (resto = 0 significa que ele é divisível)
- A entrada deve ser pelo teclado (usar input)
- Para calcular o resto, use o operador %, exemplos:

  20 % 2 = 0

  11 % 2 = 1

  11 % 3 = 2

# Números primos (2)

- Melhorar algoritmo do desafio anterior
- Não precisamos calcular o resto de números maiores que a metade do número sendo testado