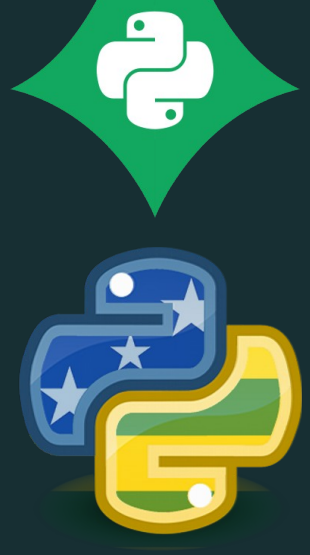
Orientação a Objetos com Python e UML



George Mendonça
@george_mendonca
www.tihardcore.wordpress.com

Agenda

1. Super Mini Tutorial
2. UML
3. Classe minimalista
4. Atributo, escopo, instância e método
5. classmethod
6. Herança
7. Outros conceitos de OO
8. TDD com Python
9. Comunidades

Super Mini Tutorial Python

- Linguagem de programação orientada a objetos
- Funcional
- Alto nível
- Interpretada
- Semântica dinâmica integrada
- Visualmente atraente e de fácil interpretação e aprendizado
- Reduzindo custo de desenvolvimento e manutenção
- Para desenvolvimento de aplicativos, para web e mobile
- Acessível

Super Mini Tutorial Python

- Simplista – Mínimo esforço do programador sobre o esforço computacional
- Clara e objetiva
- Não utiliza *begin*/*end*, chaves ou qualquer outro delimitador
- Identação obrigatória
- Tipagem dinâmica e forte
- Software Livre
- Criador: Guido van Rossum
- Mantido pela Python Software Foundation

Super Mini Tutorial Python

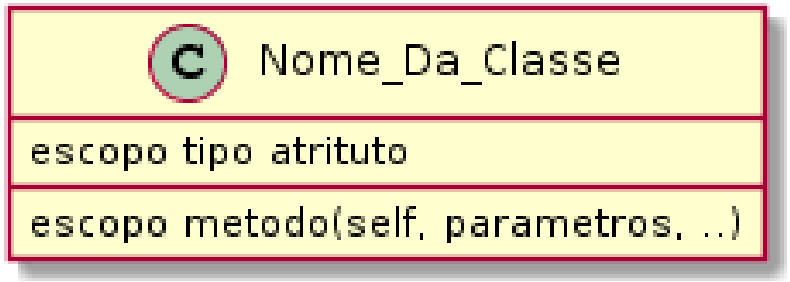
```
# coding: utf-8
# Comentáriode 1 linha
'''
Comentário de várias
linhas
'''
# Variáveis
salario = 12000.0

# Função
def minhaFuncao(s):
    print('Meu salário: %.2f')%(s)

# Chamando a função e imprimindo salario
minhaFuncao(salario)

if salario > 5000.00:
    print('Sênior')
```

UML

- **Unified Modeling Language**
  - Linguagem de modelagem para o desenvolvimento de software
  - Que permite representar um sistema de forma padronizada
  - Não uma metodologia, mas auxlilia na visualização do modelo do projeto de software
  - Na construção de projetos orientado a objetos, facilita a comunicação entre os objetos

UML

- **Diagrama de classes**

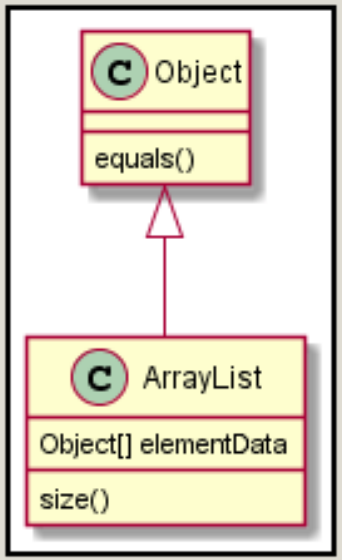


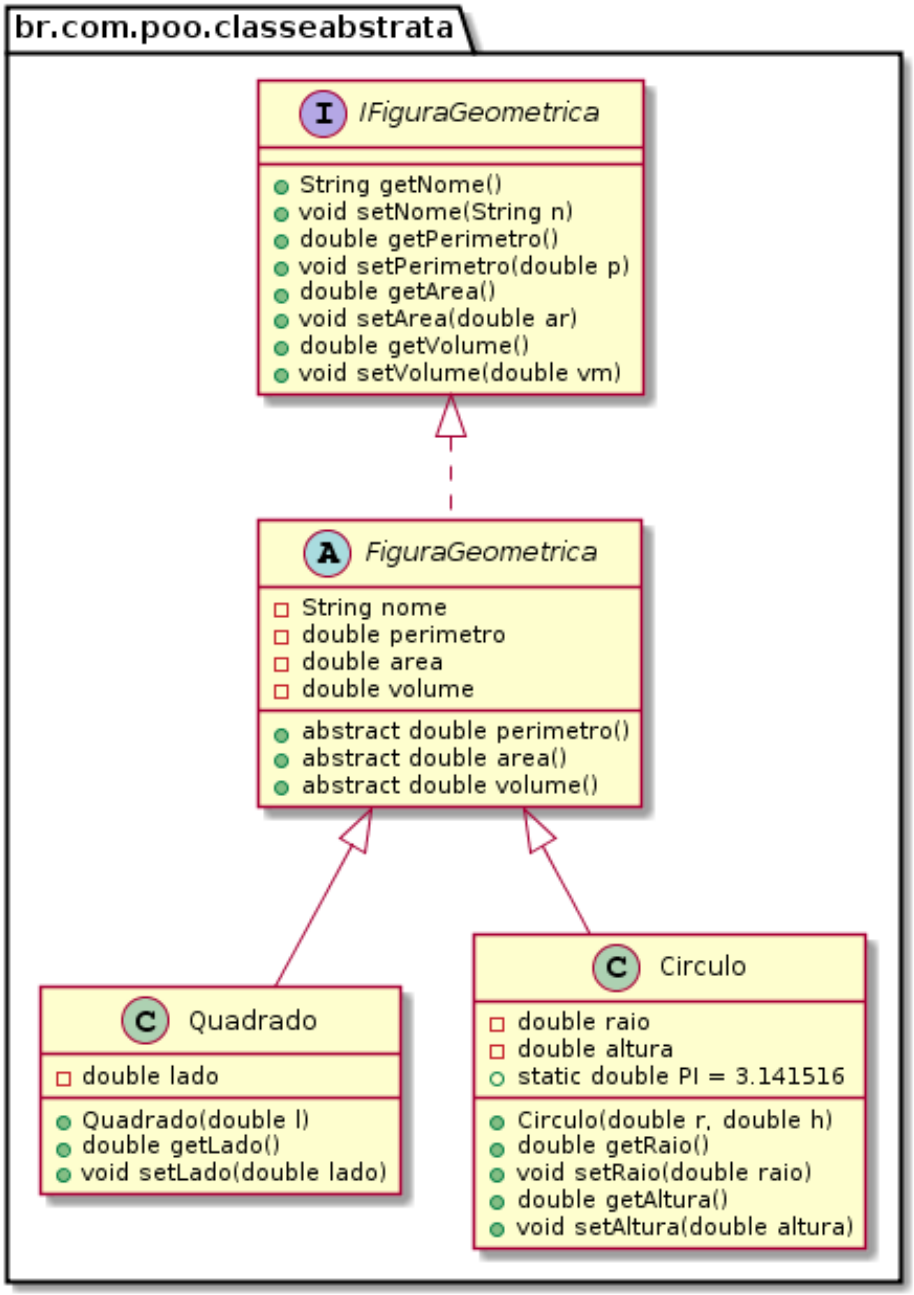
UML

- Diagrama de classes
- Solftware Livre utilizad
  - Plant UML
  - GPLv3
  - http://plantuml.com

```
@startuml
Object <|-- ArrayList

Object : equals()
ArrayList : Object[] elementData
ArrayList : size()

@enduml
```

UML por notação !





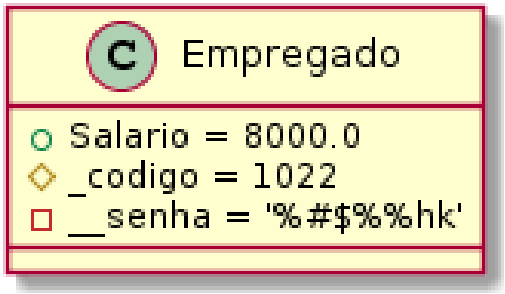
Classe minimalista em Python

- Vamos começar com a classe mais simples em Python:


```
class Minima:
    pass
```

Atributo, escopo, instância e método

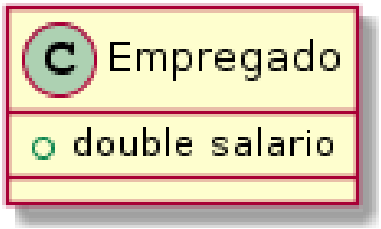
- Um atributo é uma variável da classe




```
class Empregado:
    Salario = 8000.0 # PÚBLICO
    _codigo = 1022 # PROTEGIDO
    __senha = '%#$%%hk' # PRIVADO
```

Atributo, escopo, instância e método

- Um atributo é uma variável da classe




```
class Empregado:
    salario = 8000.0

if __name__ == '__main__':
    obj = Empregado()
    print (obj.salario)
```

Console:

```
8000.0
```

Atributo, escopo, instância e método

- O **atributo** agora é uma **variável da classe**

salario é um atributo privado ! Só tem acesso na classe e não pode ser acessado fora dela!

```
class Empregado:
    __salario = 8000.0

    def getSal(self):
        return self.__salario

if __name__ == '__main__':
    obj = Empregado()
    print (obj.getSal())
```

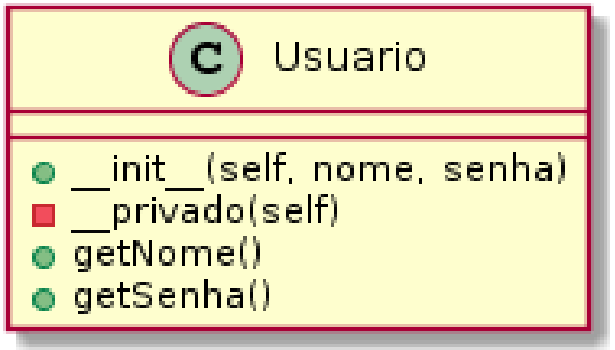
Acesso via método

Salario é Recuperado pelo método **getSal()**

Console:

```
8000.0
```

Atributo, escopo, instância e método




```
class Usuario():

    def __init__(self, nome, senha):
        self._nome = nome
        self.__senha = senha

    def __privado(self):
        print ('Só acessa a classe')

    def getNome(self):
        return self._nome

    def getSenha(self):
        return self.__senha

if __name__ == '__main__':
    us = Usuario('George', '%$@%FHGFG5457')
    # print(us._nome()) # print(us._nome())
        # => TypeError: 'str' object is not callable
    # print(us.__senha) # print(us.__senha)
        # => AttributeError: 'Usuario' object has no attribute '__senha'
    # print(us.__privado())
        # => AttributeError: 'Usuario' object has no attribute '__privado'
    print(us.getNome())
    print(us.getSenha())
```

Construtor

Atrutos protegido e privado respectivamente

Método privado

Console:

```
George
%$@%FHGFG5457
```

Transformando um Método em Método da Classe - Classmethod

- (**método**) -> convertido em um **método** da classe (estáico)
  - Um método de classe recebe a classe como primeiro argumento implícito
  - Assim como um método de instância recebe a instância

```
class ClasseMetodo:
    def f(self):
        print('Massa !')
    f = classmethod(f)

if __name__ == '__main__':
    ClasseMetodo.f()
```

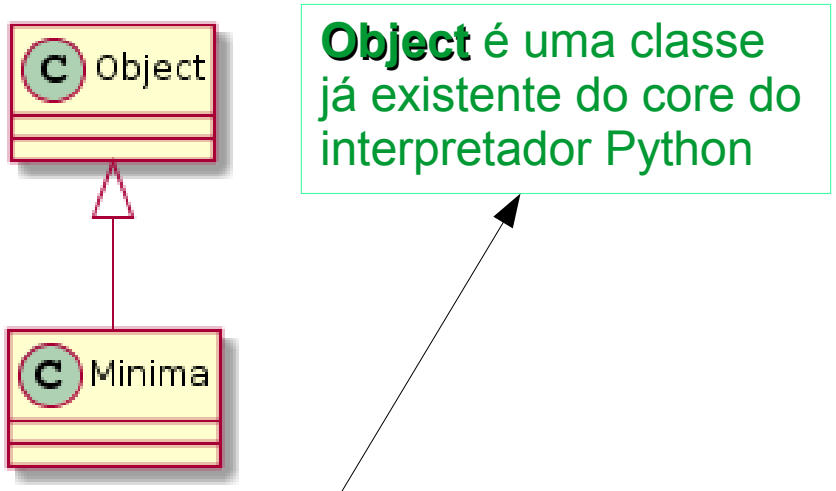
```
TypeError: unbound method f() must be called with MetodoClasse instance as first argument (got nothing instead)
```

Console:

```
Massa!
```

Herança em Python

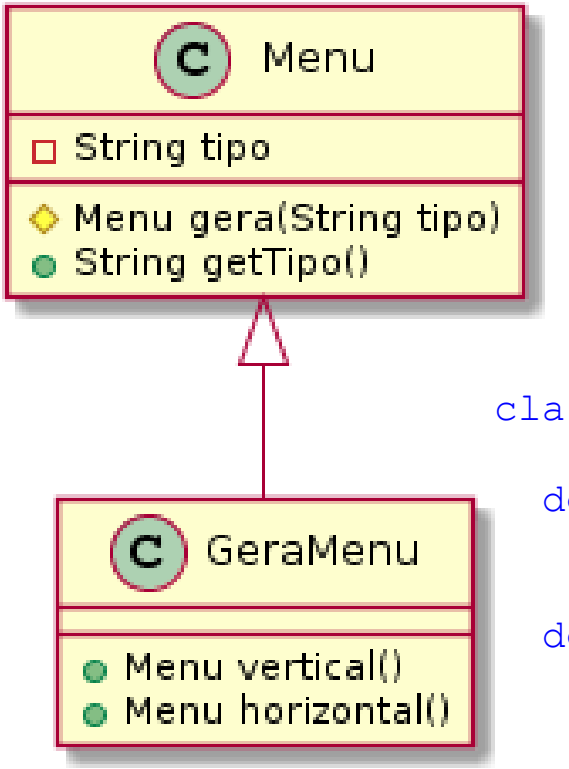
- Criando uma subclasse – **Herança**




```
class Minima(object):
    pass
```

- A herança também é utilizada para a propagação de **metaclasses** (ver artigo Metaclasses em Python)

Herança em Python




```
class Menu():
    __tipo = ""

    def gera(self, tipo):
        menu = Menu()
        menu.__tipo = tipo
        return menu

    def getTipo(self):
        return self.__tipo

class GeraMenu(Menu):

  def vertical(self):
    return Menu.gera(self, '> Menu Vertical').getTipo()

  def horizontal(self):
    return Menu.gera(self, '> Menu Horizontal').getTipo()

if __name__ == '__main__':
    menu = GeraMenu()
    print(menu.vertical())
    print(menu.horizontal())
```

Console:

```
> Menu Vertical
> Menu Horizontal
```

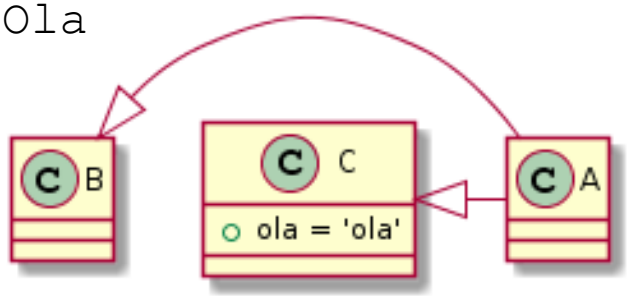
Outros conceitos de OO em Python

- **Herança Múltipla**

```
class C(object):
    ola = 'ola'

class B(object): pass

class A(B, C):
    pass

if __name__ == '__main__':
    a = A()
    print(a.ola)
```

Console:

```
Ola
```

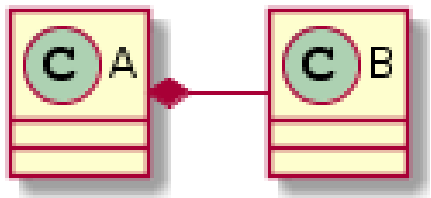


- **Agragação e composição**

```
class B(object): pass

class A(object):
    def __init__(self, b):
        self.b = b

if __name__ == '__main__':
    b = B()
    a = A(b)
    print(a.b)
```

Console:

```
<__main__.B object at 0x7f808be4a9d0>
```

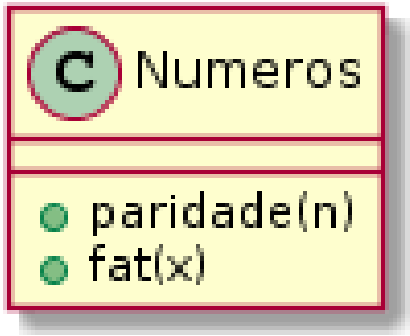


TDD – Um resumo só bre Test Driven Development

- **Desenvolvimento Guiado por Teste**
  - Kent Beck (XP)
  - Qualidade
  - Confiabilidade
  - Prática: Teste priemeiro, implementação depois
- TDD com Python é Moleza!
- Existem vários frames, mas utilizaremos **unittest** para um exemplo bem básico! (mais utilizado)

TDD – Um resumo só bre Test Driven Development

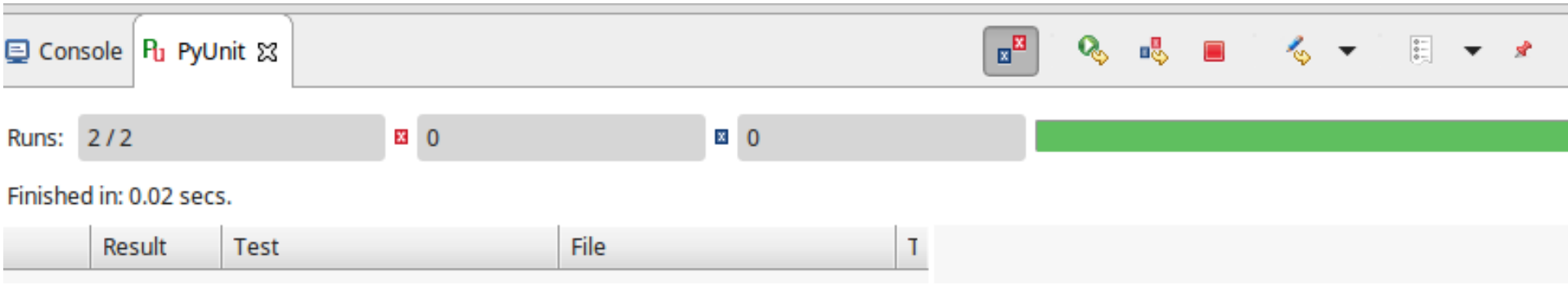
```
import unittest
from pyoo.tdd.Numeros import Numeros

class TesteNumeroPrimo(unittest.TestCase):

    def testeParidade(self):
        p = Numeros()
        self.assertEqual(p.paridade(0), 0)
        self.assertEqual(p.paridade(1), 1)
        self.assertEqual(p.paridade(2), 0)
        self.assertEqual(p.paridade(3), 1)

    def testeFatorial(self):
        p = Numeros()
        self.assertEqual(p.fat(0), 1)
        self.assertEqual(p.fat(1), 1)
        self.assertEqual(p.fat(2), 2)
        self.assertEqual(p.fat(3), 6)
        self.assertEqual(p.fat(4), 24)
        self.assertEqual(p.fat(5), 120)

if __name__ == '__main__':
    unittest.main()
```




```
class Numeros():

    def paridade(self, n):
        return n % 2

    def fat(self, x):
        f = 1
        for i in range(x):
            f *= (i + 1)
        return f
```

TDD – Um resumo só bre Test Driven Development

- A barrinha VERDE ...



Comunidade

- **Grypy-GO** – Grupo de Usuários Python de Goiás
  - https://www.facebook.com/groups/grupygo
  - https://telegram.me/grupygo
- **Grypy-DF** – Grupo de Usuários Python do Distrito Federal
  - https://www.facebook.com/groups/grupydf
  - https://telegram.me/grupydf
  - http://df.python.org.br
- **Python Brasil**
  - https://www.facebook.com/pythonbrasil
  - https://www.facebook.com/groups/python.brasil

Valeu galera, até o XIV FGSL!!!!!!

Obrigado!

Slides e código fonte:

Git Lab: https://gitlab.com/gepds/poo/tree/master/eventos/XII-FGSL

SlideShare:
http://www.slideshare.net/georgemendonca/orientao-a-objetos-com-python-e-uml-xiii-fgs

Conditions of use

You can use this OpenOffice Impress template for your personal, educational and business presentations.

The copyright statement we require you to include when you use our material is:

© Copyright Showeet.com

http://www.showeet.com

Contact: Showeet@ymail.com

creative commons

With the use of this free template you accept the following use and license conditions.

You are free:

To Share — to copy, distribute and transmit the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works — You may not alter, transform, or build upon this work.

For any distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page: http://www.showeet.com/terms-of-use/

Any of the conditions can be waived if you get permission from showeet.com

In no event shall Showeet.com be liable for any indirect, special or consequential damages arising out of or in connection with the use of the template, diagram or map.

http://creativecommons.org/licenses/by-nd/3.0/