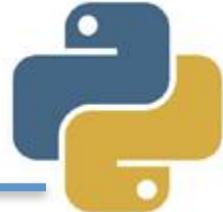
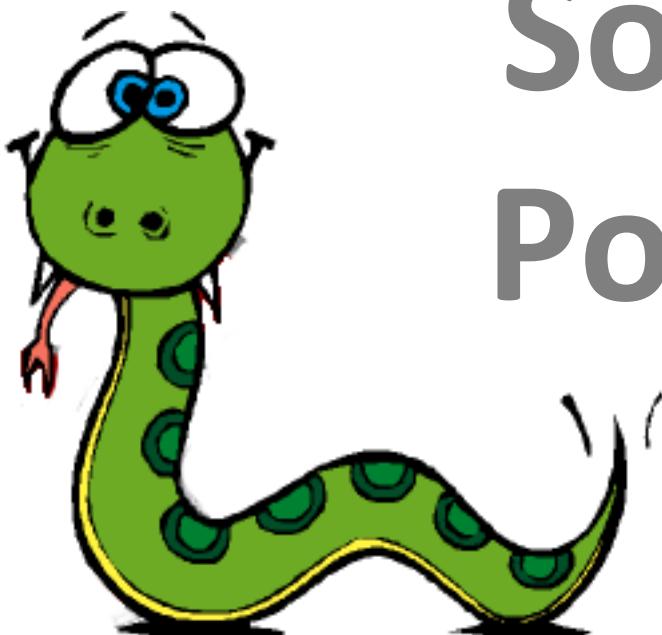


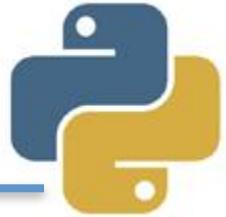
Capítulo 05



Herança, Sobrescrita e Polimorfismo



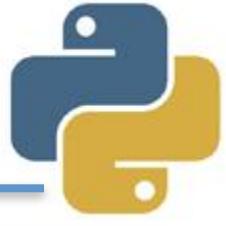
Márcio Palheta, M.Sc.
marcio.palheta@gmail.com



Apresentação

- Programador desde 2000
- Professor de programação desde 2009
- Mestre pelo ICOMP/UFAM – 2013
- Fundador da Buritech – 2014
- Doutorando pelo ICOMP/UFAM
- Pesquisador das áreas: Banco de Dados, Recuperação da Informação, Big Data, Mineração de dados e Aprendizado de Máquina

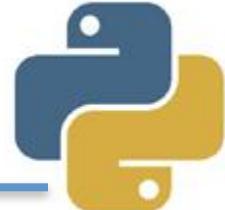




Agenda

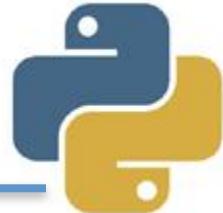
- Revisão da aula anterior
 - Herança entre classes
 - Sobrescrita de métodos
 - Polimorfismo pythônico
 - Decorator @classmethod
 - Herança múltipla
-

Revisão



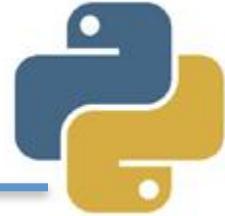
- Encapsulamento
 - Atributos privados
 - Decorators @property e @atrib.setter
 - Old style versus new style class
 - Atributos e métodos estáticos
 - Decorator @staticmethod
-

O que temos por aqui?

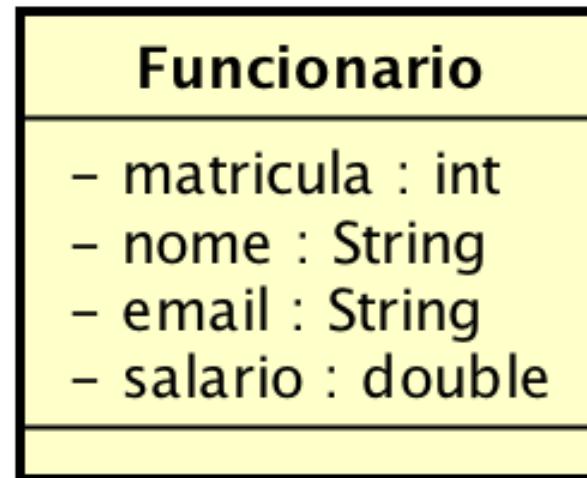


- O que é e como devemos usar a Herança?
 - O papel da classe object
 - Sobrescrita de métodos
 - Como implementar a Herança múltipla ?
 - Qual a ordem de execução de métodos na herança múltipla?
-

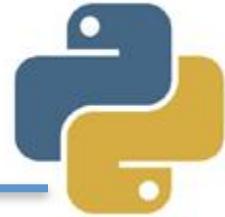
E o tempo passa...



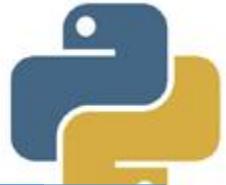
- Com o passar do tempo, a nossa instituição financeira viu a necessidade de controlar os **dados de Funcionários**:



Exercício 1: Funcionário



- No arquivo **model.py**, crie a classe **Fucionario**, conforme slide anterior
 - A **matrícula** deve ser **auto-incremento** e **somente leitura**
 - Crie um **construtor** que receba **nome**, **email** e **salário**, com seus respectivos **valores padrão**
-



Exercício 1: Funcionário

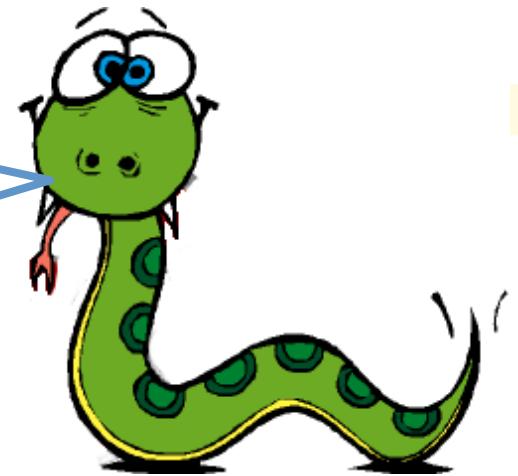
```
model.py x
  Funcionario __init__()
1  # -*- coding: UTF-8 -*-
2  # model.py
3
4  class Funcionario(object):
5      __contador = 0
6
7      def __init__(self, nome=None, email=None, salario=0.0):
8          # incrementa o contador
9          Funcionario.__contador += 1
10         # matricula auto-incremento
11         self.__matricula = Funcionario.__contador
12         self.nome = nome
13         self.email = email
14         self.salario = salario
15
16     @property
17     def matricula(self):
18         return self.__matricula
19
```

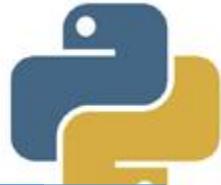
Exercício 1: Funcionário



```
model.py x
  Funcionario __init__()
1  # -*- coding: UTF-8 -*-
2  # model.py
3
4  class Funcionario(object):
5      __contador = 0
6
7      def __init__(self, nome=None, email=None, salario=0.0):
8          # incrementa o contador
9          Funcionario.__contador += 1
10         # matrícula auto-incremento
11
12         # Atributos da classe Funcionario
13         # nome, email e salario
14
15         # Matrícula é gerada automaticamente
16
17         def matricula(self):
18             return self.__matricula
19
```

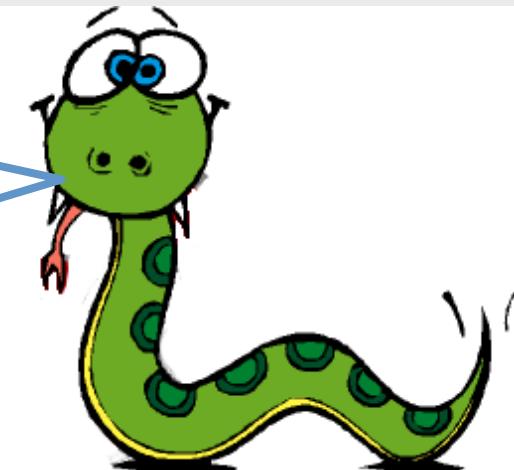
Contador para criação automática da matrícula





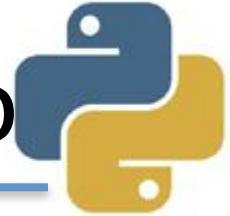
Exercício 1: Funcionário

Atribuição e **getter** para matrícula



```
model.py x
1   Funcionario __init__()
2
3
4
5
6
7   def __init__(self, nome=None, email= None):
8       # incrementa o contador
9       Funcionario.__contador += 1
10      # matricula auto-incremento
11      self.__matricula = Funcionario.__contador
12      self.nome = nome
13      self.email = email
14      self.salario = salario
15
16      @property
17      def matricula(self):
18          return self.__matricula
```

Exercício 2: teste_funcionario



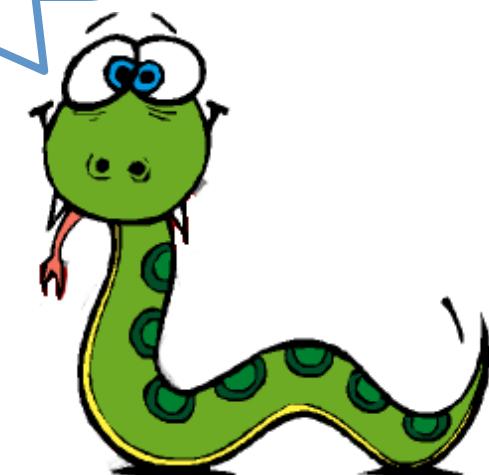
```
1 # -*- coding: UTF-8 -*-
2 # teste_funcionario.py
3
4 from model import Funcionario
5
6 funcionario = Funcionario(salario=1000)
7 funcionario2 = Funcionario(salario=500)
8 funcionario3 = Funcionario()
9
10 print 'Funcionario 1\nMatricula:', funcionario.matricula
11 print 'Salário:', funcionario.salario
12
13 print '\nFuncionario 2\nMatricula:', funcionario2.matricula
14 print 'Salário:', funcionario2.salario
15
16 print '\nFuncionario 3\nMatricula:', funcionario3.matricula
17 print 'Salário:', funcionario3.salario
18
```

Exercício 2: t

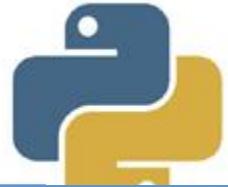
Criação de Funcionários



```
1 # -*- coding: UTF-8 -*-
2 # teste_funcionario.py
3
4 from model import Funcionario
5
6 funcionario = Funcionario(salario=1000)
7 funcionario2 = Funcionario(salario=500)
8 funcionario3 = Funcionario()
9
10 print 'Funcionario 1\nMatricula:', funcionario.matricula
11 print 'Salário:', funcionario.salario
12
13 print '\nFuncionario 2\nMatricula:', funcionario2.matricula
14 print 'Salário:', funcionario2.salario
15
16 print '\nFuncionario 3\nMatricula:', funcionario3.matricula
17 print 'Salário:', funcionario3.salario
18
```



Resultado no console



```
Run teste_funcionario
/Library/Frameworks/Python.framework/Versions/2.7
Funcionario 1
Matricula: 1
Salário: 1000

Funcionario 2
Matricula: 2
Salário: 500

Funcionario 3
Matricula: 3
Salário: 0.0

Process finished with exit code 0
```

Resultado no console



```
Run teste_funcionario
/Library/Frameworks/Python.framework/Versions/2.7
Funcionario 1
Matricula: 1
Salário: 1000

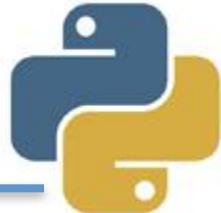
Funcionario 2
Matricula: 2
Salário: 500

Funcionario 3
Matricula: 3
Salário: 0.0
```

Tudo **funcionou** corretamente

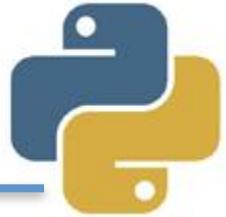


Process finished with exit code 0

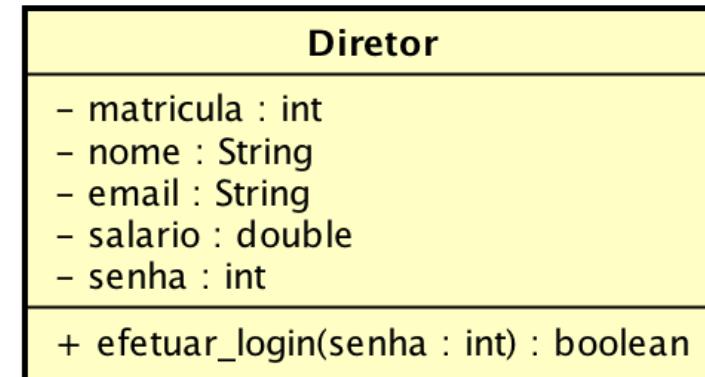
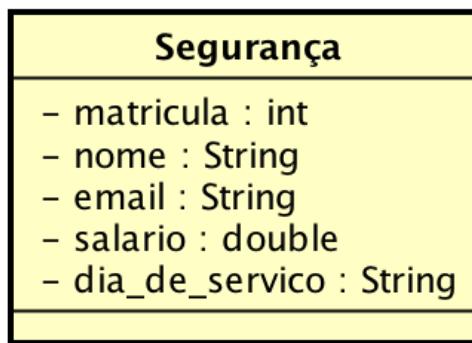
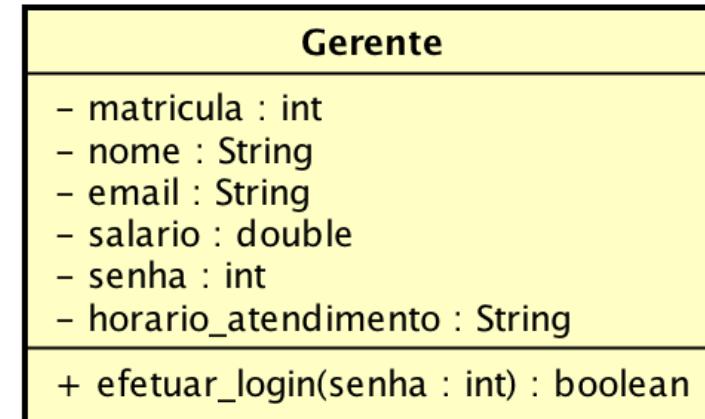
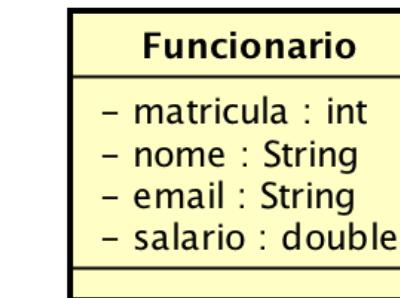


Mas a vida...

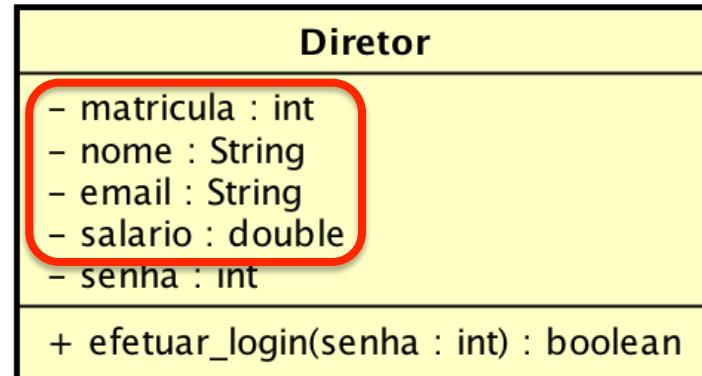
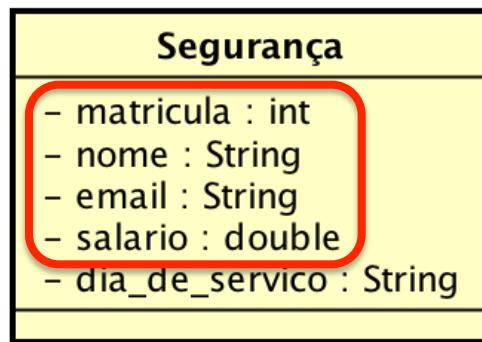
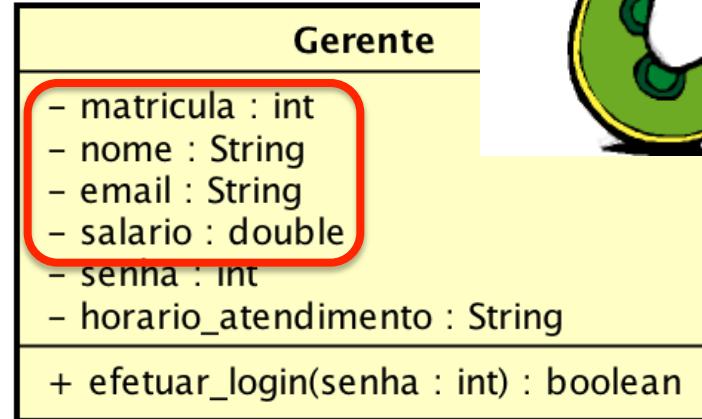
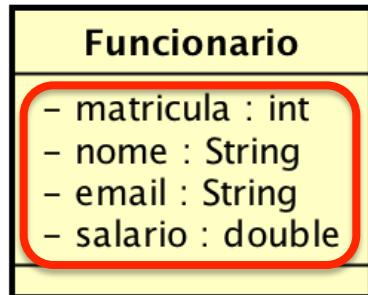
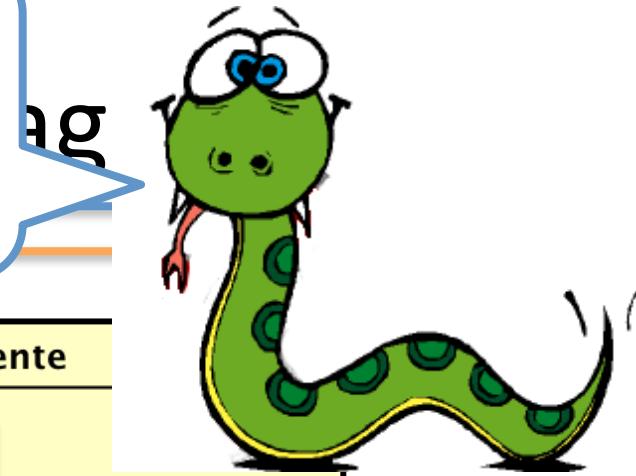
- verificou-se que alguns **tipos** de funcionários possuem **dados peculiares**:
 - Diretor** e **Gerente** possuem uma **senha para autenticação** nos sistemas
 - Diretores** só atendem em **horário específico**
 - Segurança** não trabalha todos os dias
-



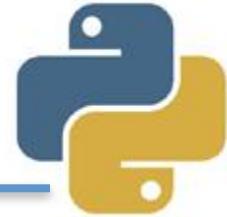
Voltando à modelagem



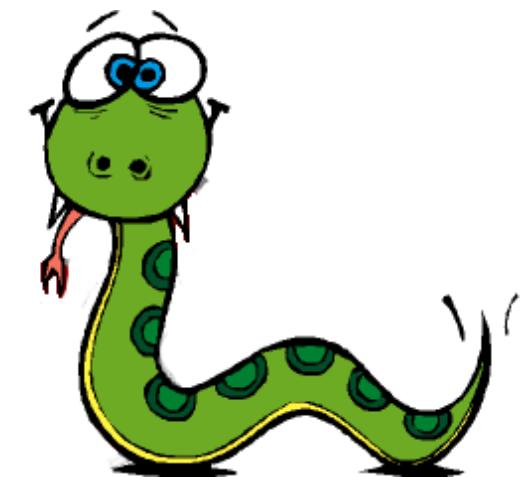
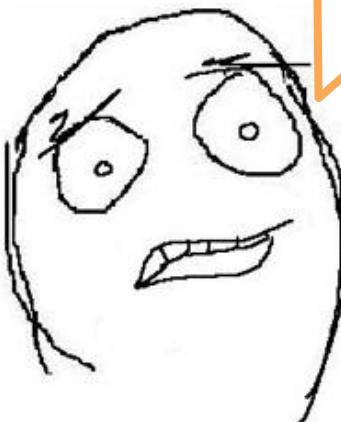
Parece que alguns atributos são comuns a todos os tipos de funcionário



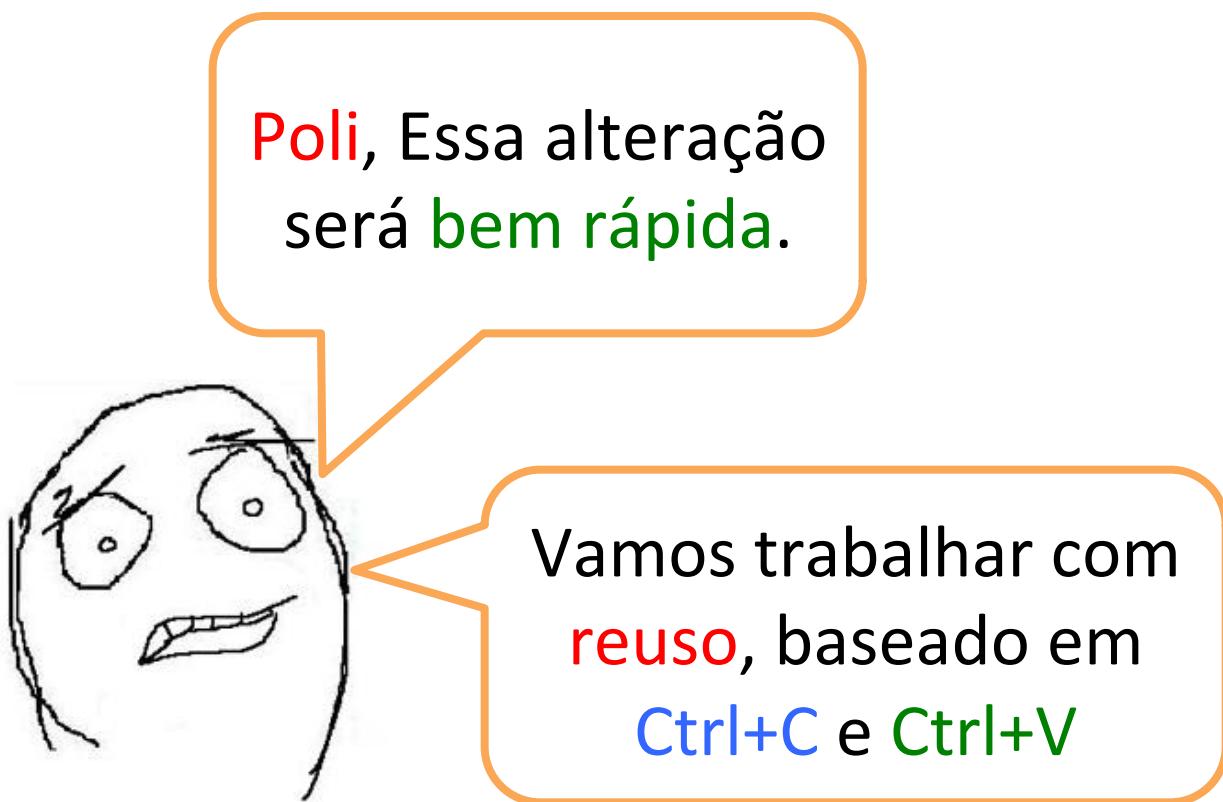
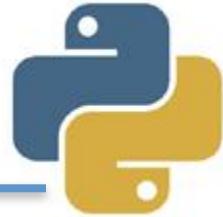
Pensando em reuso



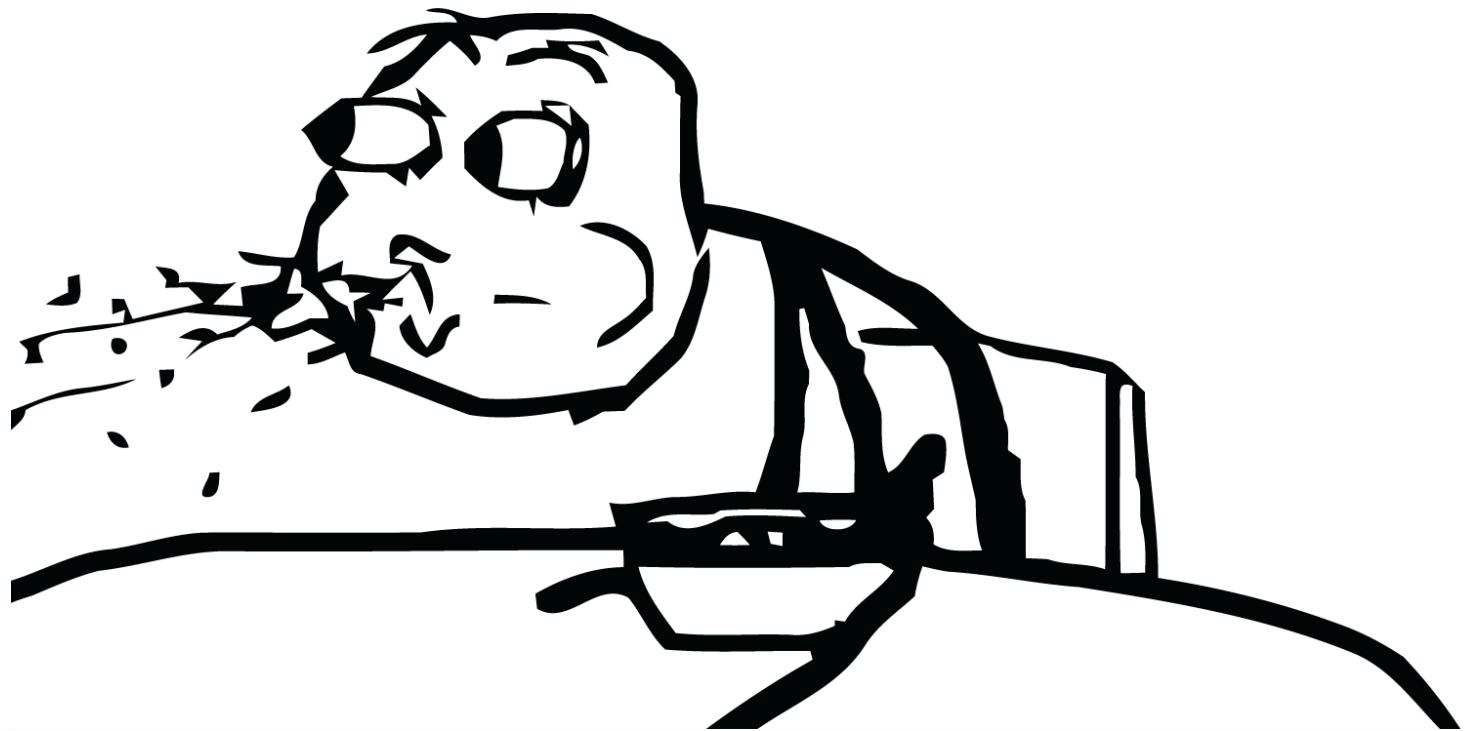
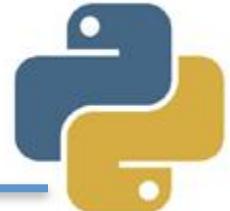
Poli, Essa alteração
será bem rápida.



Pensando em reuso



Pensando em reuso



Pensando em reuso

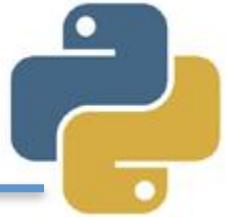


Poli, Essa alteração
será bem rápida.

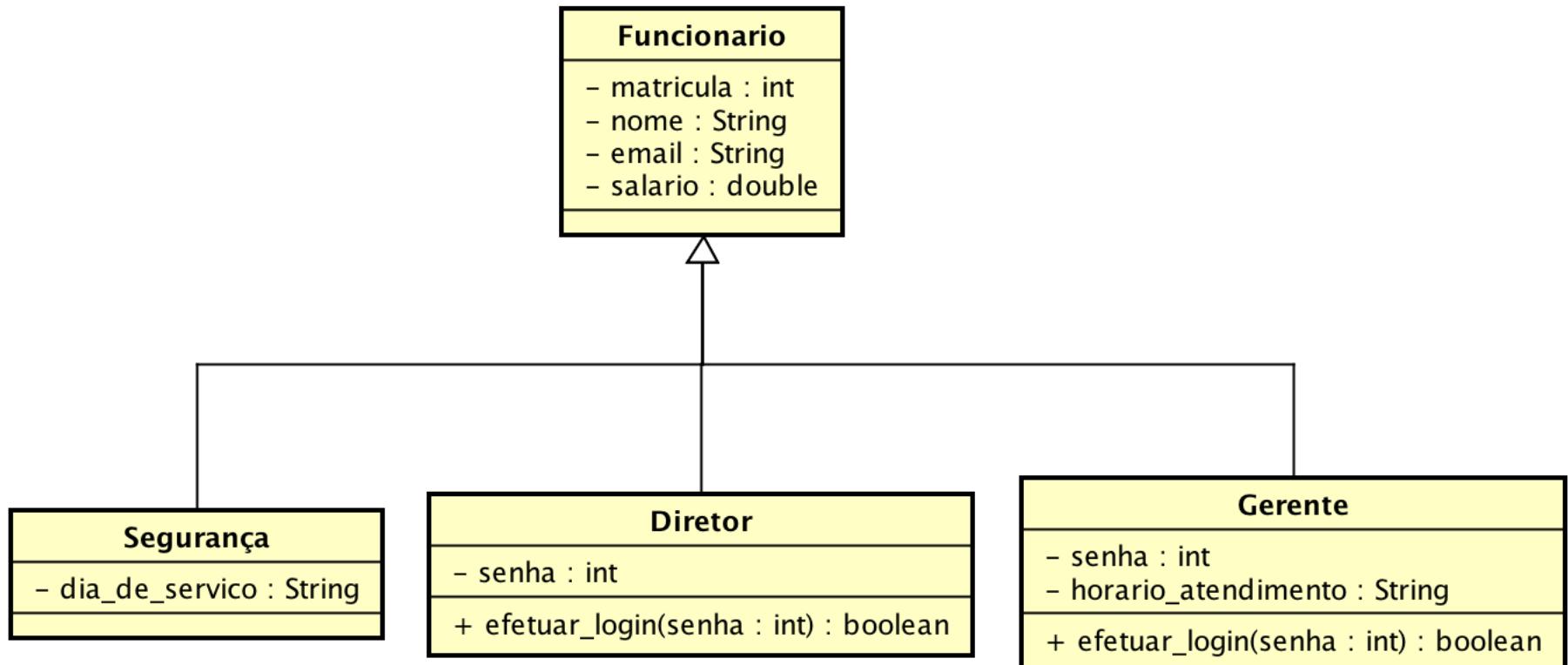
Vamos trabalhar com
reuso, baseado em
Ctrl+C e Ctrl+V

Morfismo, acho que
não é bem esse o
conceito de reuso

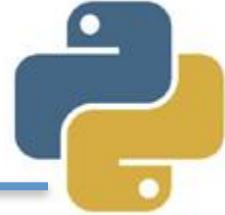




Herança

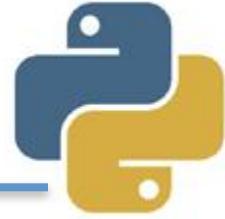


Herança



- Em Orientação a Objetos, podemos marcar as classes Segurança, Gerente e Diretor como filhas da classe Funcionário
 - Assim, as classes filhas passariam a herdar métodos e atributos da classe pai
 - Mas as filhas só possuem acesso direto aos componentes públicos da classe pai
-

Exercício 3: classe Seguranca

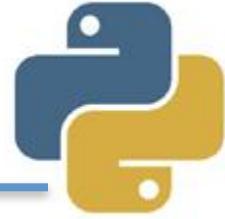


- Em `model.py`, crie a classe **Seguranca**
- **Seguranca** deve ser **filha** da **Funcionario**
- Precisamos **invocar** o **construtor** do pai

A screenshot of a code editor window titled "model.py". The code defines a class named "Seguranca" which inherits from "Funcionario". The constructor __init__ calls the constructor of the parent class Funcionario and initializes a specific attribute dia_servico.

```
model.py
1 Seguranca
20 class Seguranca(Funcionario):
21     def __init__(self, dia_servico=None):
22         #Inicializa os atributos da classe Pai
23         Funcionario.__init__(self)
24         self.dia_servico = dia_servico
25
```

Exercício 3: classe Seguranca

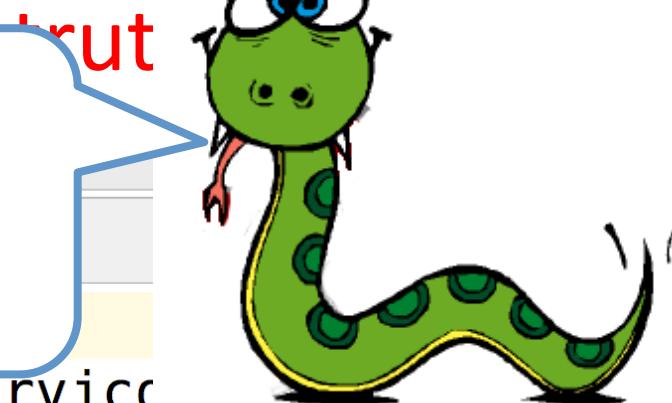


- Em `model.py`, crie a classe **Seguranca**
- **Seguranca** deve ser **filha** da **Funcionario**

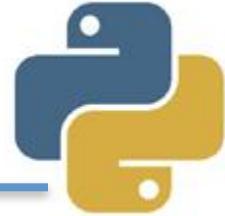
- P

Forma **mais simples** de
chamada a **métodos** da
superclasse

```
20
21     def __init__(self, dia_servico):
22         #Inicializa os atributos da classe Pai
23         Funcionario.__init__(self)
24         self.dia_servico = dia_servico
25
```



Exercício 4: teste_heranca



- Crie o arquivo `teste_heranca.py`
 - Importe `Funcionario` e `Seguranca`
 - Crie os objetos `funcionario(salario=1000)` e `seguranca(dia_servico="seg,qua,sex")`
 - Imprima os dados dos objetos
 - Veja o valor de `matrícula` e `salário` do `segurança`
-

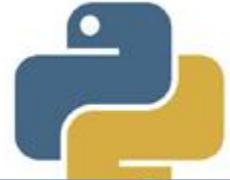


Exercício 4: teste_heranca

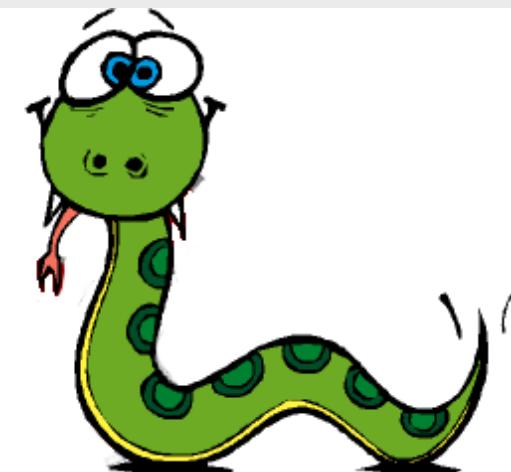
```
model.py x teste_heranca.py x

1 # -*- coding: UTF-8 -*-
2 # teste_heranca.py
3
4 from model import Funcionario
5 from model import Seguranca
6
7 funcionario = Funcionario(salario=1000)
8 seguranca = Seguranca(dia_servico="seg,qua,sex")
9
10 print 'Funcionario\nMatricula:', funcionario.matricula
11 print 'Salário:', funcionario.salario
12
13 print '\nSeguranca\nMatricula:', seguranca.matricula
14 print 'Salário:', seguranca.salario
15 print 'Dias de serviço:', seguranca.dia_servico
16
```

Exercício 4: teste_heranca



Importação das classes



```
model.py x teste_heranca.py x

1 # -*- coding:
2 # teste_heranca.py

3
4 from model import Funcionario
5 from model import Seguranca
6
7 funcionario = Funcionario(salario=1000)
8 seguranca = Seguranca(dia_servico="seg,qua,sex")
9
10 print 'Funcionario\nMatricula:', funcionario.matricula
11 print 'Salário:', funcionario.salario
12
13 print '\nSeguranca\nMatricula:', seguranca.matricula
14 print 'Salário:', seguranca.salario
15 print 'Dias de serviço:', seguranca.dia_servico
16
```



Exercício 4: teste_heranca

Criação de
objetos

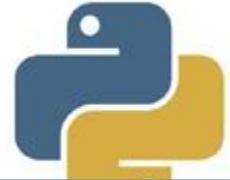


```
model.py x teste_heranca.py x

1 # -*- coding: UTF-8
2 # teste_heranca.py

3
4 from model import Funcionario
5 from model import Seguranca
6
7 funcionario = Funcionario(salario=1000)
8 seguranca = Seguranca(dia_servico="seg,qua,sex")
9
10 print 'Funcionario\nMatricula:', funcionario.matricula
11 print 'Salário:', funcionario.salario
12
13 print '\nSeguranca\nMatricula:', seguranca.matricula
14 print 'Salário:', seguranca.salario
15 print 'Dias de serviço:', seguranca.dia_servico
16
```

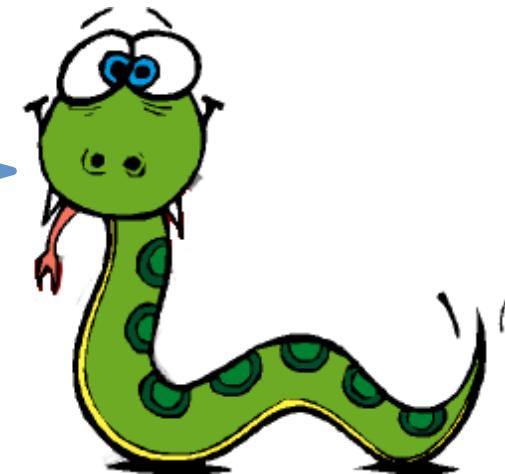
Exercício 4: teste_heranca



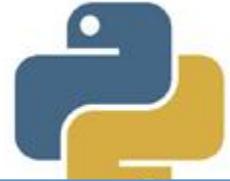
model.py x teste_heranca.py x

```
1 # -*- coding: UTF-8
2 # teste_heranca.py
3
4 from model import Funcionario
5 from model import Seguranca
6
7 funcionario = Funcionario(salario=1000)
8 seguranca = Seguranca(dia_servico="seg")
9
10 print 'Funcionario\nMatricula:', funcionario.matricula
11 print 'Salário:', funcionario.salario
12
13 print '\nSeguranca\nMatricula:', seguranca.matricula
14 print 'Salário:', seguranca.salario
15 print 'Dias de serviço:', seguranca.dia_servico
16
```

Impressão
do conteúdo



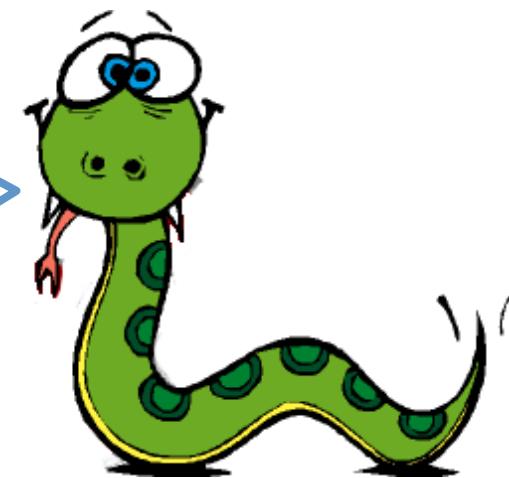
Exercício 4: teste_heranca



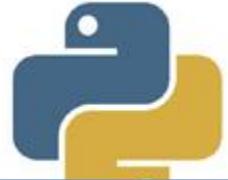
model.py x teste_heranca.py x

```
1 # -*- coding: UTF-8 -*-
2 # teste_heranca.py
3
4 from model import Funcionario
5 from model import Seguranca
6
7 funcionario = Funcionario()
8 seguranca = Seguranca()
9
10 print 'Funcionario\nMatrícula:', funcionario.matricula
11 print 'Salário:', funcionario.salario
12
13 print '\nSegurança\nMatrícula:', seguranca.matricula
14 print 'Salário:', seguranca.salario
15 print 'Dias de serviço:', seguranca.dia_servico
16
```

Usando a
matrícula
herdada



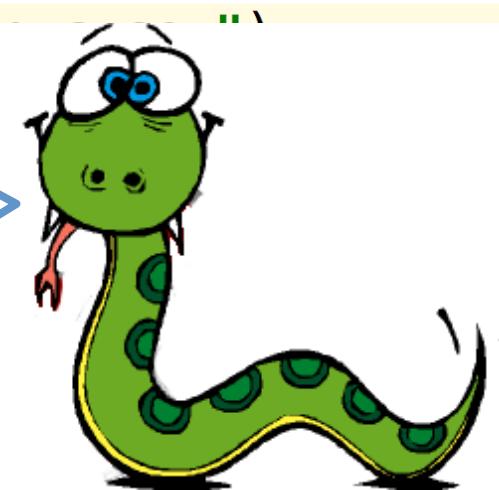
Exercício 4: teste_heranca

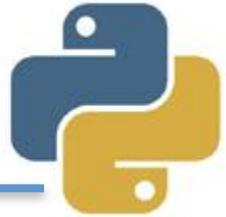


```
model.py x teste_heranca.py x

1 # -*- coding: UTF-8 -*-
2 # teste_heranca.py
3
4 from model import Funcionario
5 from model import Seguranca
6
7 funcionario = Funcionario('João', 1234567890)
8 segurança = Seguranca('João', 1234567890)
9
10 print 'Funcionario'
11 print 'Salário:', funcionario.salario
12
13 print '\nSegurança\nMatrícula:', segurança.matricula
14 print 'Salário:', segurança.salario
15 print 'Dias de serviço:', segurança.dias_de_servico
```

Usando o
salário
herdado



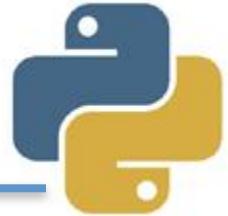


Resultado no console

```
Run teste_heranca
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/funcionarios.py:10: DeprecationWarning: 
  print("Matricula: ", self.matricula)
  print("Salário: ", self.salario)
  print("Dias de serviço: ", self.dias_de_servico)
DeprecationWarning: 
  print("Matricula: ", self.matricula)
  print("Salário: ", self.salario)
  print("Dias de serviço: ", self.dias_de_servico)
Seguranca
Matricula: 2
Salário: 0.0
Dias de serviço: seg,qua,sex

Process finished with exit code 0
```

Resultado no console

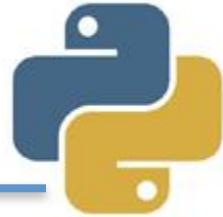


```
Run teste_heranca
/Library/Frameworks/Python framework/Version
Func.
Matr.
Salá
Segurança
Matrícula: 2
Salário: 0.0
Dias de serviço: seg,qua,sex
Process finished with exit code 0
```

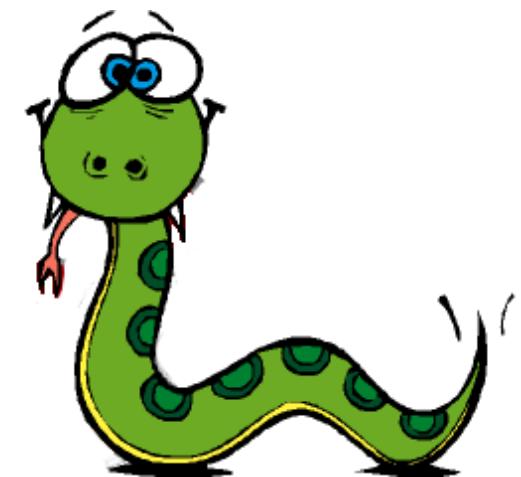
Os atributos herdados se comportaram como o esperado

A cartoon illustration of a green snake with large eyes and a smiling mouth, positioned to the right of the speech bubble.

Pensando em reuso



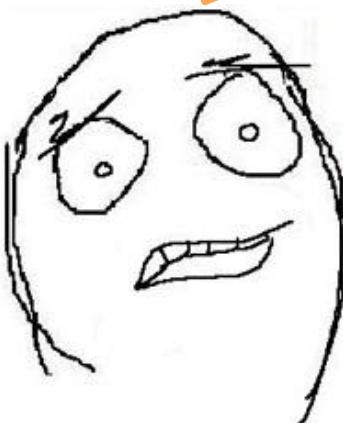
Poli, como eu faria se quisesse passar o salário do segurança no construtor?



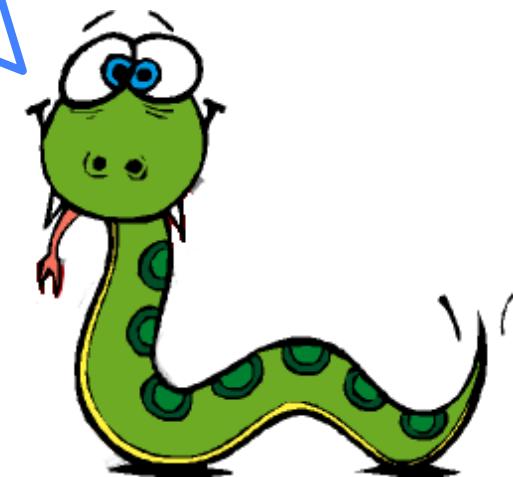
Pensando em reuso



Poli, como eu faria se quisesse passar o salário do segurança no construtor?



Morfismo, você deveria receber o argumento na classe filha...



Pensando em reuso

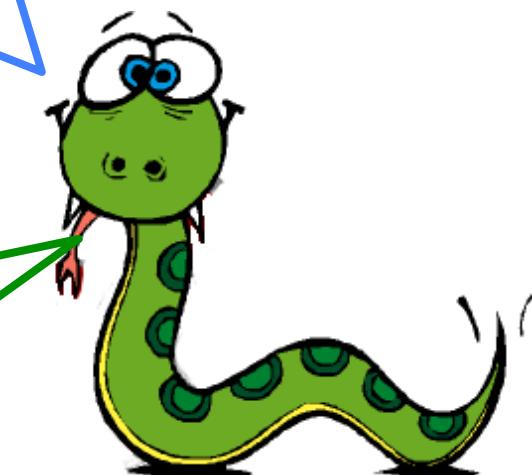


Poli, como eu faria se quisesse passar o salário do segurança no construtor?

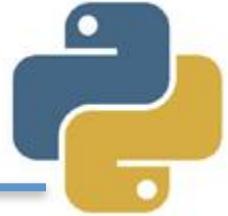


Morfismo, você deveria receber o argumento na classe filha...

... e enviar-lo para a classe pai



Exercício 5: salário inicial



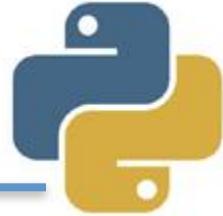
- Altere o **construtor** da classe **Segurança**
- Receba o **salário** do segurança e o envie para o **construtor** da classe **pai**

A screenshot of a code editor showing a Python file named 'model.py'. The code defines a class 'Seguranca' that inherits from 'Funcionario'. The constructor '__init__' is overridden to initialize attributes specific to 'Seguranca' and call the constructor of 'Funcionario' with the 'salario' parameter.

```
model.py x teste_heranca.py x
Seguranca __init__()

19
20 class Seguranca(Funcionario):
21     def __init__(self, dia_servico=None, salario=None):
22         #Inicializa os atributos da classe Pai
23         Funcionario.__init__(self, salario=salario)
24         self.dia_servico = dia_servico
25
```

Exercício 5: salário inicial



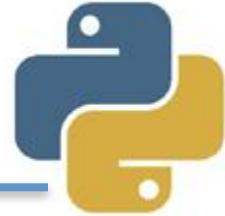
- Altere o **construtor** da classe **Segurança**
- Receba o **salário** do segurança e o envie para o **construtor** da classe **pai**

A screenshot of a code editor showing Python code related to inheritance. The code defines a class 'Seguranca' that inherits from 'Funcionario'. The constructor '__init__' is overridden to accept a 'salario' parameter, which is then passed to the parent class's constructor. Two specific lines of code are highlighted with red boxes: the parameter 'salario=None' in the child class's constructor and the call to 'Funcionario.__init__' with 'salario=salario'.

```
model.py x teste_heranca.py x
Seguranca __init__()

19
20 class Seguranca(Funcionario):
21     def __init__(self, dia_servico=None, salario=None):
22         #Inicializa os atributos da classe Pai
23         Funcionario.__init__(self, salario=salario)
24         self.dia_servico = dia_servico
25
```

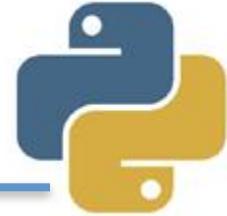
Exercício 6: teste do salário



- Altere o script **teste_heranca.py**
- Informe o **salário inicial** do **segurança** e **execute** o script novamente

```
1 # -*- coding: UTF-8 -*-
2 # teste_heranca.py
3
4 from model import Funcionario
5 from model import Seguranc
6
7 funcionario = Funcionario(salario=1000)
8 seguranc = Seguranc(dia_servico="seg,qua,sex", salario=800)
9
```

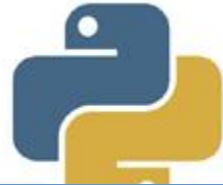
Exercício 6: teste do salário



- Altere o script **teste_heranca.py**
- Informe o **salário inicial** do **segurança** e **execute** o script novamente

```
1 # -*- coding: UTF-8 -*-
2 # teste_heranca.py
3
4 from model import Funcionario
5 from model import Seguranc
6
7 funcionario = Funcionario(salario=1000)
8 seguranc = Seguranc(dia_servico="seg,qua.sex", salario=800)
9
```

Resultado no console



Run teste_heranca

```
/Library/Frameworks/Python.framework/Ve
Funcionario
Matricula: 1
Salário: 1000

Seguranca
Matricula: 2
Salário: 800
Dias de serviço: seg,qua,sex
```

Process finished with exit code 0

Resultado no console

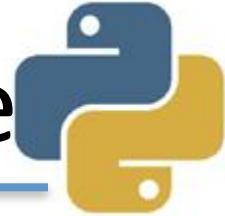


```
Run teste_heranca
/Library/Frameworks/Python.framework/Ve
Funcionar
Matricula
Salário:
Seguranc
Matricula: 2
Salário: 800
Dias de serviço: seg,qua,sex
Process finished with exit code 0
```

Agora o
segurança possui
um salário inicial



Exercício 7: Diretor e Gerente



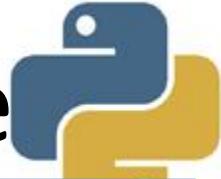
- No arquivo `model.py`, crie as classes **Diretor** e **Gerente**
 - As duas devem receber o salário como argumento **opcional** de seus respectivos métodos de **inicialização**
 - Ambas precisam do método de **validação** **efetuar_login()**
-

Exercício 7: Diretor e Gerente



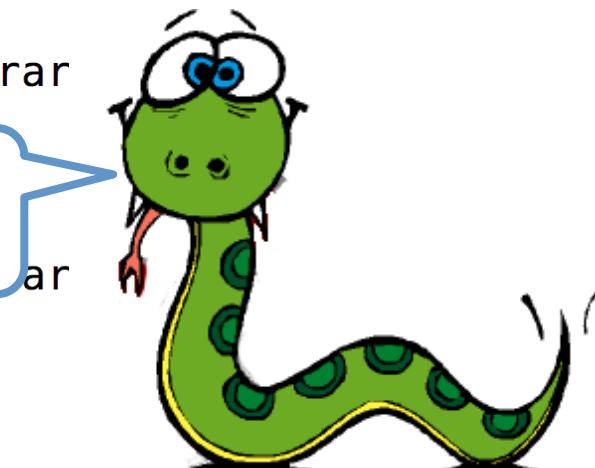
```
model.py x
Diretor
30 class Diretor(Funcionario):
31     def __init__(self, senha=None, salario=None):
32         #Forma mais correta para invocar a super classe
33         super(Diretor, self).__init__(salario=salario)
34         self.senha = senha
35
36     def efetuar_login(self, senha):
37         return self.senha == senha
38
39 class Gerente (Funcionario):
40     def __init__(self, senha=None, horario_atendimento=None,
41                  salario=None):
42         super(Gerente, self).__init__(salario=salario)
43         self.senha = senha
44         self.horario_atendimento = horario_atendimento
45
46     def efetuar_login(self, senha):
47         return self.senha == senha
48
```

Exercício 7: Diretor e Gerente



```
model.py x
Diretor
30 class Diretor(Funcionario):
31     def __init__(self, senha=None, salario=None):
32         #Forma mais correta para invocar a super classe
33         super(Diretor, self).__init__(salario=salario)
34         self.senha = senha
35
36     def efetuar_login(self, senha):
37         return self.senha == senha
38
39 class Gerente (Funcionario):
40     def __init__(self, senha=None, horario=None,
41                  salario=None):
42         super().__init__(salario=salario)
43         self.senha = senha
44
45     def efetuar_login(self, senha):
46         return self.senha == senha
```

A classe **Diretor**

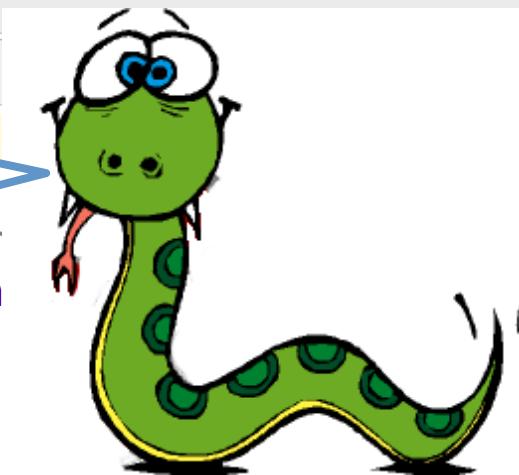


Exercício 7: Diretor e Gerente

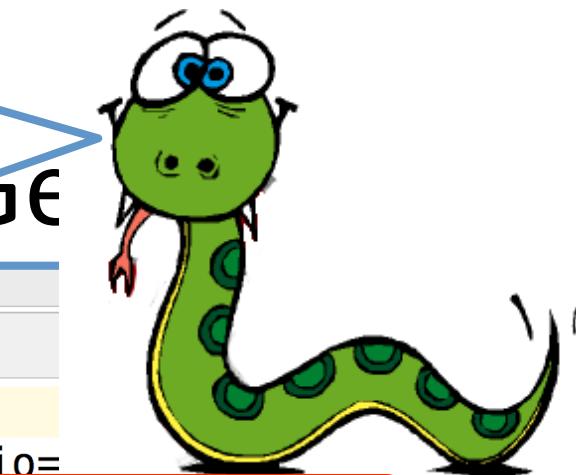


```
model.py x
Diretor
30 class Diretor(Funcionario):
31     def __init__(self, senha):
32         #Força a senha
33         super(Diretor, self).__init__(senha)
34         self.senha = senha
35
36     def efetuar_login(self, senha):
37         return self.senha == senha
38
39 class Gerente(Funcionario):
40     def __init__(self, senha=None, horario_atendimento=None,
41                  salario=None):
42         super(Gerente, self).__init__(salario=salario)
43         self.senha = senha
44         self.horario_atendimento = horario_atendimento
45
46     def efetuar_login(self, senha):
47         return self.senha == senha
48
```

A classe **Gerente**

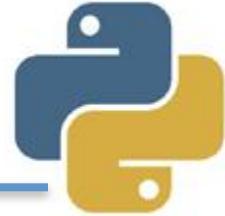


Forma mais correta para invocar métodos da classe pai, por não informarmos o seu nome



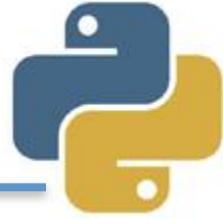
```
31     def __init__(self, senha=None, salario=0):
32         #Forma mais correta para invocar a super classe
33         super(Diretor, self).__init__(salario=salario)
34         self.senna = senna
35
36     def efetuar_login(self, senha):
37         return self.senha == senha
38
39 class Gerente(Funcionario):
40     def __init__(self, senha=None, horario_atendimento=None,
41                  salario=None):
42         super(Gerente, self).__init__(salario=salario)
43         self.senna = senna
44         self.horario_atendimento = horario_atendimento
45
46     def efetuar_login(self, senha):
47         return self.senha == senha
48
```

Exercício 8: teste herança



- No arquivo `teste_heranca.py`, importe as classes **Diretor** e **Gerente**
 - **Crie** objetos das classes e **inicialize** seus atributos
 - **Imprima** os valores dos atributos dos novos objetos
-

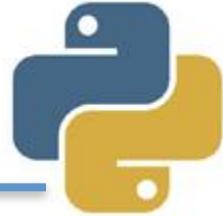
Exercício 8: teste herança



- Atualização do script de teste com **importação** e **criação** de objetos

```
1 # -*- coding: UTF-8 -*-
2 # teste_heranca.py
3
4 from model import Funcionario
5     from model import Seguranca
6     from model import Diretor
7     from model import Gerente
8
9     funcionario = Funcionario(salario=1000)
10    seguranca = Seguranca(dia_servico="seg,qua.sex", salario=800)
11    diretor = Diretor(1234, 5000.0)
12    gerente = Gerente(5678, '10:00 - 12:00', 3500.0)
13
```

Exercício 8: teste herança



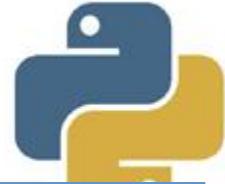
- Atualização do script de teste com importação e criação de objetos

Faça essas alterações em teste_heranca.py

```
1 # -*- coding: utf-8 -*-
2 # teste_heranca.py
3
4 from model import Funcionario
5 from model import Seguranc
6 from model import Diretor
7 from model import Gerente
8
9 funcionario = Funcionario(salario=1000)
10 seguranca = Seguranc(dia_servico="seg,qua.sex", salario=800)
11 diretor = Diretor(1234, 5000.0)
12 gerente = Gerente(5678, '10:00 - 12:00', 3500.0)
```



Exercício 8: continuação



```
21 print '\nDiretor\nMatrícula:', diretor.matricula
22 print 'Salário:', diretor.salario
23 print 'Senha:', diretor.senha
24 if diretor.efetuar_login(1111):
25     print 'Diretor logado'
26 else:
27     print 'Falha de autenticação do Diretor'
28
29
30 print '\nGerente\nMatrícula:', gerente.matricula
31 print 'Salário:', gerente.salario
32 print 'Senha:', gerente.senha
33 print 'Atendimento:', gerente.horario_atendimento
34 if gerente.efetuar_login(5678):
35     print 'Gerente logado'
36 else:
37     print 'Falha de autenticação do Gerente'
```

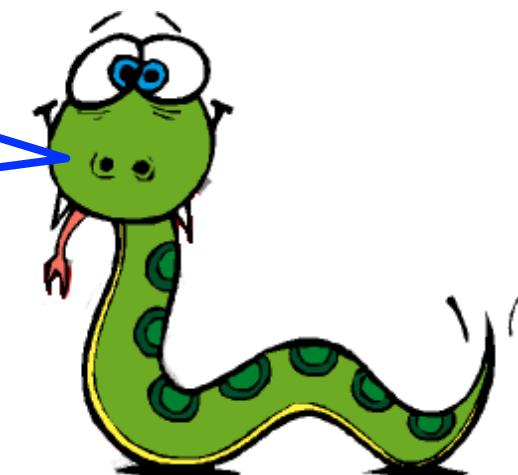


Exercício 8: continuação

```
21 print '\nDiretor\nMatrícula:', diretor.matricula  
22 print 'Salário:', diretor.salario  
23 print 'Senha:', diretor.senha  
24 if diretor.efetuar_login(1111):  
25     print 'Diretor logado'  
26 else:  
27     print 'Falha de autenticação do Diretor'  
28  
29         gerente.matricula  
30         io  
31 Faça essas alterações para  
32 exibir dados do Diretor  
33         gerente.efetuar_login(5555):  
34             print 'Gerente logado'  
35 else:  
36     print 'Falha de autenticação'
```

Faça essas alterações para
exibir dados do Diretor

gerente.matricula
io



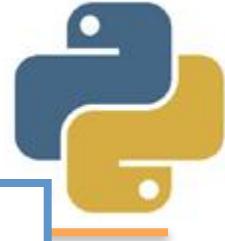


Faça essas alterações para
exibir dados do **Gerente**

```
21 print senha, diretor.senha
22 if diretor.efetuar_login(1111):
23     print 'Diretor logado'
24 else:
25     print 'Falha de autenticação do Diretor.
26
27
28
29 print '\nGerente\nMatricula:', gerente.matricula
30 print 'Salário:', gerente.salario
31 print 'Senha:', gerente.senha
32 print 'Atendimento:', gerente.horario_atendimento
33 if gerente.efetuar_login(5678):
34     print 'Gerente logado'
35 else:
36     print 'Falha de autenticação do Gerente'
```



Resultado no console



Diretor

Matricula: 3

Salário: 5000.0

Senha: 1234

Falha de autenticação do Diretor

Gerente

Matricula: 4

Salário: 3500.0

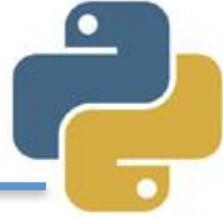
Senha: 5678

Atendimento: 10:00 – 12:00

Gerente logado

Process finished with exit code 0

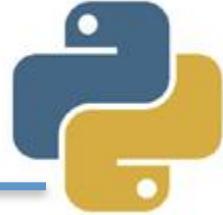
built-in function: super()



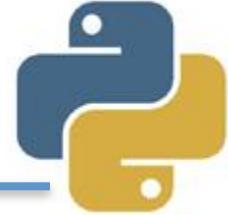
- “[Super is used to] *return a proxy object that delegates method calls to a parent or sibling class of type. This is useful for accessing inherited methods that have been overridden in a class. The search order is same as that used by getattr() except that the type itself is skipped.*”

[Documentação oficial do Python](#)

built-in function: super()

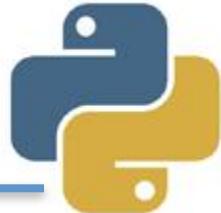


- Usamos `super()` em dois casos:
 - Caso 1: em `herança simples` para nos referirmos à classe pai, sem nomeá-la
 - Caso 2: pode ser usado em `tempo de execução` para herança múltipla ou colaborativa. Não é possível em LP que suportam apenas herança simples.
-



Chamada a métodos herdados

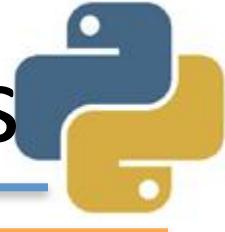
- Forma mais **simples**:
 - SuperClass.method(instance, args)
 - Formas mais **corretas**:
 - Python **2**:
 - super(SubClass, instance).method(args)
 - Python **3**:
 - super().method_name(args)
-



Mas a vida...

- A alta gerência da instituição decidiu **pagar um bônus** para seus **funcionários**
 - Cada funcionário deverá receber como bônus o valor do **salário + 20%**
 - Neste cenário, um funcionário que recebe **R\$ 1.000,00**, ganha um bônus de **R\$ 1.200,00**
-

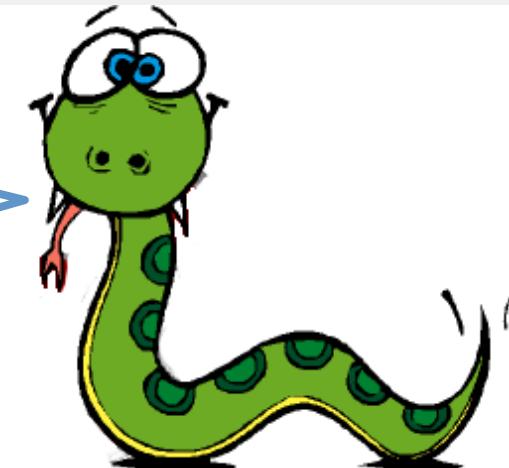
Exercício 9: getter para bônus



Função bonus()

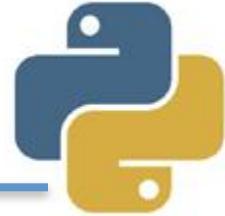
Na classe **Funcionário**, crie
a **@property** para o **bônus**

```
return self.__mat
```



```
@property
def bonus(self):
    return self.salario * 1.2
```

Exercício 10: teste do bônus

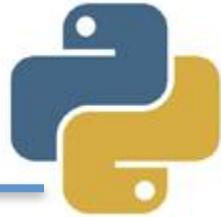


- No arquivo **teste_heranca.py**, inclua os testes de **bônus** dos **funcionários**

```
#Facilitando o cálculo
funcionario.salario = 1000.0
seguranca.salario = 1000.0
diretor.salario = 1000.0
gerente.salario = 1000.0

print '\nSalário e Bônus:'
print 'funcionário', funcionario.salario, funcionario.bonus
print 'segurança', seguranca.salario, seguranca.bonus
print 'gerente', gerente.salario, gerente.bonus
print 'diretor', diretor.salario, diretor.bonus
```

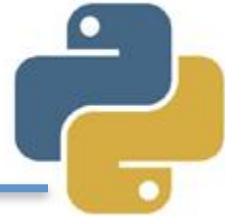
Resultado no console



- Veja que as **classes filhas** **herdaram** o método de **retorno de bônus**

```
Run teste_heranca
Sal.: Bônus:
funcionário 1000.0 1200.0
seguranca    1000.0 1200.0
gerente      1000.0 1200.0
diretor       1000.0 1200.0
Process finished with exit code 0
```

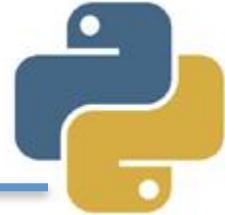
Exercício 11: bônus de 50%



Funcionario bonus()

```
@property  
def matricula(self):  
    return self.__matricula  
  
@property  
def bonus(self):  
    return self.salario * 1.5
```

Exercício 11: bônus de 50%



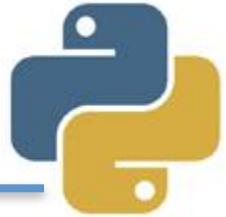
Funcionario.bonus()

Altere o valor do bônus de
1.2 para 1.5

```
return self.__ma
```



```
@property
def bonus(self):
    return self.salario * 1.5
```



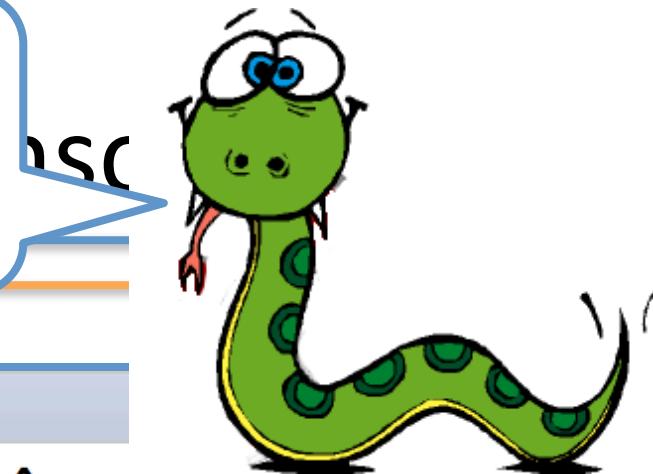
Resultado no console

Run teste_heranca

	Sal.:	Bônus:
funcionário	1000.0	1500.0
segurança	1000.0	1500.0
gerente	1000.0	1500.0
diretor	1000.0	1500.0

Process finished with exit code 0

Conseguimos resolver o problema dos diretores...



Run teste_heranca

	Sal.:	Bônus:
funcionário	1000.0	1500.0
segurança	1000.0	1500.0
gerente	1000.0	1500.0
diretor	1000.0	1500.0

Process finished with exit code 0

Mas agora todo
funcionário recebe 50% de bônus

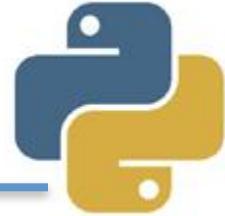


Run teste_heranca

	Sal.:	Bônus:
funcionário	1000.0	1500.0
segurança	1000.0	1500.0
gerente	1000.0	1500.0
diretor	1000.0	1500.0

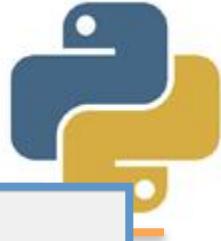
Process finished with exit code 0

Sobrescrita de métodos



- Nossa solução **resolveu** um problema, mas **criou outro**
 - Alterando o método **bonus()** da classe **Funcionario**, pela herança, todas as classes **filhas** são **afetadas**
 - Precisamos **sobrescrever** o método na classe **Diretor**
-

Exercício 12: bônus de 20%



```
Funcionario bonus()
```

```
class Funcionario(object):
    __contador = 0

    def __init__(self, nome=None, email=N
                 @property
                 def matricula(self):...
                 @property
                 def bonus(self):
                     return self.salario * 1.2
```

Exercício 12: bônus de 20%



Fun

cla

Volte o valor do **bônus** do
funcionário para 20%

```
def __init__(self, nome)
```

```
@property
```

```
def matricula(self): ...
```

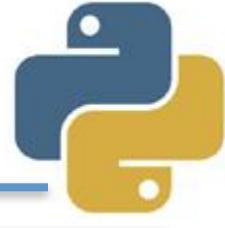


```
@property
```

```
def bonus(self):
```

```
    return self.salario * 1.2
```

Exercício 13: bônus de 50%



```
Diretor bonus()
```

```
class Diretor(Funcionario):
    def __init__(self, senha=None, salario=None):
        Funcionario.__init__(self, salario=salario)
        self.senha = senha

    def efetuar_login(self, senha):
        return self.senha == senha

@property
def bonus(self):
    return self.salario * 1.5
```

Exercício 13: bônus de 50%



```
Diretor bonus()
```

```
class Diretor(Funcionario):
    def __init__(self, senha=None, s
        Funcionario __init__(self, s
            )
    ):
```

Sobrescreva o método
bônus na classe Diretor



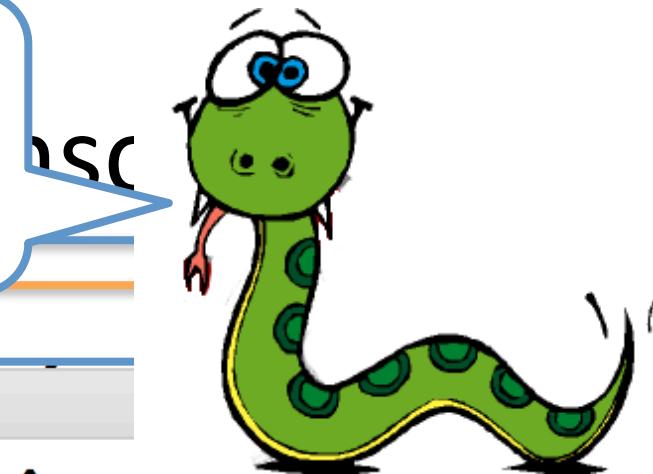
```
@property
def bonus(self):
    return self.salario * 1.5
```

Resultado no console



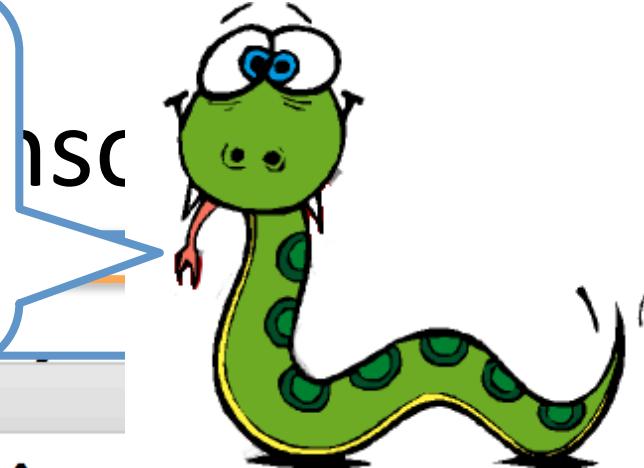
```
Run teste_heranca
▶ ⬆ ⬇ ⏹ ⏷ ⏸ ⏹ ⏺ ⏻ ⏻ ? Sal.: Bônus:
funcionário 1000.0 1200.0
segurança 1000.0 1200.0
gerente 1000.0 1200.0
diretor 1000.0 1500.0
Process finished with exit code 0
```

Conseguimos resolver o problema dos diretores...



```
Run Python teste_heranca
Sal.: Bônus:
funcionário 1000.0 1200.0
segurança 1000.0 1200.0
gerente 1000.0 1200.0
diretor 1000.0 1500.0
Process finished with exit code 0
```

E o **bônus** dos outros funcionários voltou ao normal



Run teste_heranca

```
Sal.: Bônus:  
funcionário 1000.0 1200.0  
seguranca    1000.0 1200.0  
gerente      1000.0 1200.0  
diretor       1000.0 1500.0
```

Process finished with exit code 0

Exercício 14: Controlador



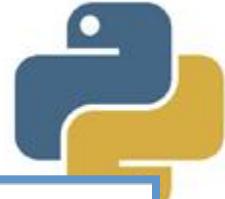
- Crie o script `control.py`, onde você vai implementar a classe `ControladorBonus`
 - Armazene uma `lista de funcionários` e exiba o `total de bônus` da lista
 - A qualquer momento, podemos `remover` ou `adicionar` um Funcionário, Segurança, Gerente ou Diretor
-

Exercício 14: Controlador



```
1 # -*- coding: UTF-8 -*-
2 # control.py
3
4 class ControleBonus(object):
5     def __init__(self):
6         self.__total = 0
7         self.__lista_funcionarios = []
8
9     @property
10    def total(self):
11        return self.__total
12
13    @property
14    def lista(self):
15        return self.__lista_funcionarios
16
```

Exercício 14: Controlador

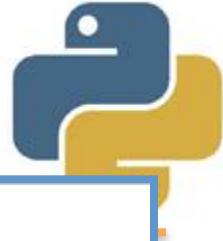


```
1 # -*- coding: UTF-8 -*-
2 # control.py
3
4 class Script
5     def control.py
6         print("Script")
7         print("control.py")
8
9         @property
10        def total(self):
11            return self._total
12
13        @property
14        def lista(self):
15            return self._lista_funcionarios
16
```

control.py

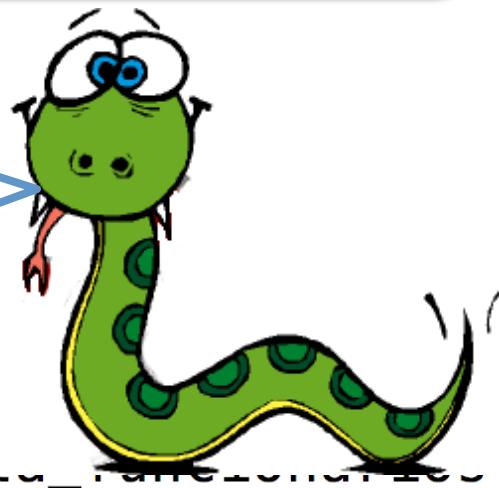


Exercício 14: Controlador

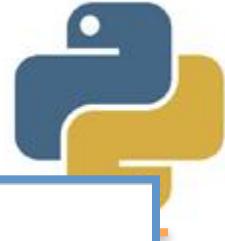


```
1 # -*- coding: UTF-8 -*-
2 # control.py
3
4 class ControleBonus(object):
5     def __init__(self):
6         self.__total = 0
7         self.__lista_funcionarios = []
8
9
10
11
12
13 @property
14 def lista(self):
15     return self.__lista_funcionarios
16
```

Definição da
classe e atributos

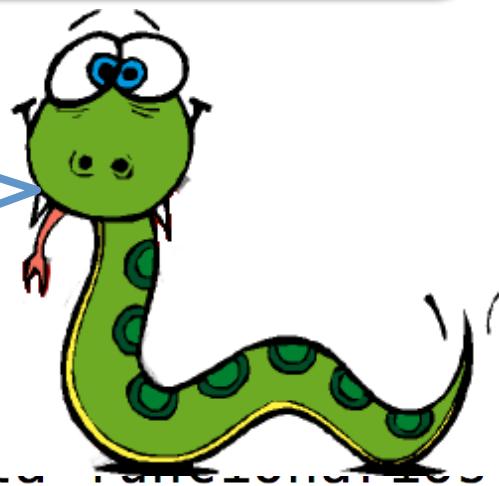


Exercício 14: Controlador

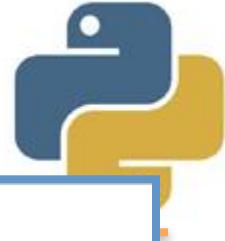


```
1 # -*- coding: UTF-8 -*-
2 # control.py
3
4 class ControleBonus(object):
5     def __init__(self):
6         self.__total = 0
7         self.__lista_funcionarios = []
8
9
10
11
12
13 @property
14 def lista(self):
15     return self.__lista_funcionarios
16
```

Definição da
classe e atributos



Exercício 14: Controlador

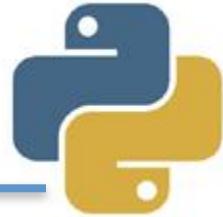


```
1  # -*- coding: UTF-8 -*-
2  #
3  #
4  class Controller:
5      def __init__(self):
6          self.__total =
7          self.__lista_funcionarios = []
8
9      @property
10     def total(self):
11         return self.__total
12
13     @property
14     def lista(self):
15         return self.__lista_funcionarios
```

Propriedade das entidades

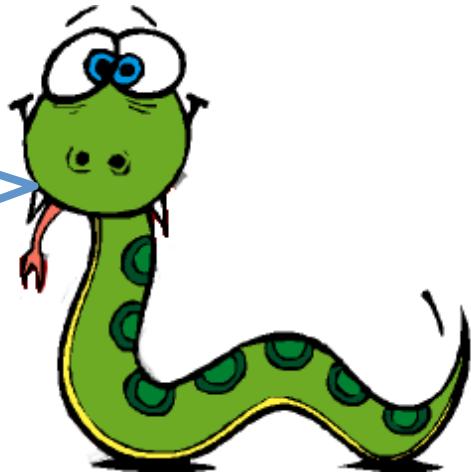


Exercício 14: continuação



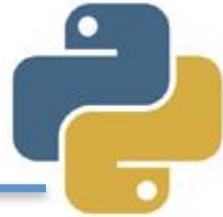
```
13     @property
14     def lista(self):
15         return self.__lista_funcionarios
16
17     @lista.setter
18     def lista(self, item):
19         #log de funcionários adicionados
20         print 'salario:', item.salario, 'bonus:', item.bonus
21         self.__lista_funcionarios.append(item)
22         self.__total += item.bonus
23
24     def excluir(self, item):
25         self.__lista_funcionarios.remove(item)
26
```

Setter para a lista de funcionários



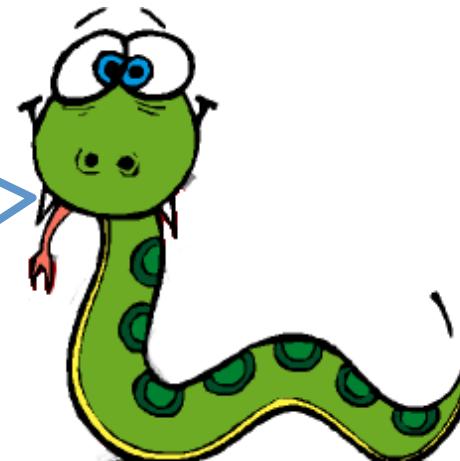
```
13
14     @property
15     def lista(self):
16         return self.__lista_fur
17
18     @lista.setter
19     def lista(self, item):
20         #log de funcionarios adicionados
21         print 'salario:', item.salario, 'bonus:', item.bonus
22         self.__lista_funcionarios.append(item)
23         self.__total += item.bonus
24
25     def excluir(self, item):
26         self.__lista_funcionarios.remove(item)
```

Exercício 14: continuação



```
13      @property  
14      def lista(self):  
15          return self.__lista_fur  
16  
17      @property  
18      def bonus(self):  
19          return self.__bonus  
20  
21      @property  
22      def funcionarios(self):  
23          return self.__funcionarios  
24  
25      def excluir(self, item):  
26          self.__lista_funcionarios.remove(item)
```

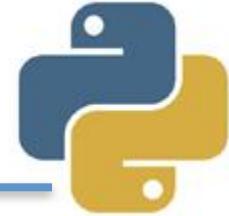
Método para exclusão de funcionários



A cartoon illustration of a green snake with large eyes and a red tongue, standing next to a speech bubble. The snake has a pattern of green circles along its body and a yellow outline.

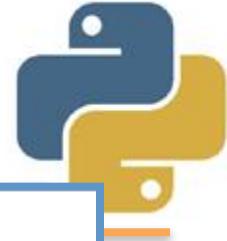
item.bonus

Exercício 15: Teste controle



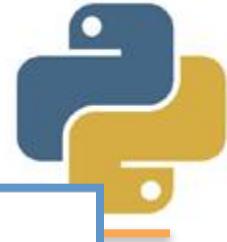
- Crie o script **teste_controle.py**
 - Crie um objeto **ControladorBonus**
 - Crie vários objetos **Funcionário** e adicione-os à **lista** do controlador
 - Imprima o **valor total** do bônus da lista
 - **Remova** objetos do controlador e imprima novamente o total
-

Exercício 15: Teste controle



```
1  # -*- coding: UTF-8 -*-
2  # teste_controlador_bonus.py
3
4  from model import Funcionario
5  from model import Seguranca
6  from model import Diretor
7  from model import Gerente
8  from control import ControleBonus
9
10 controlador = ControleBonus()
11 controlador.lista = Funcionario(salario=1000)
12 controlador.lista = Diretor(salario=1000)
13 controlador.lista = Gerente(salario=1000)
14 seguranca = Seguranca(salario=1000)
15 controlador.lista = seguranca
16
17 print 'Bônus total:', controlador.total
18 controlador.excluir(seguranca)
19 print 'Bônus total:', controlador.total
20
```

Exercício 15: Teste controle



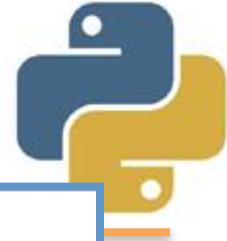
```
1 # -*- coding: UTF-8 -*-
2 # teste_controlador_bonus.py
3
4 from model import Funcionario
```

Edição do arquivo de testes:
teste_controlador_bonus.py

```
10 controlador = ControleBonus()
11 controlador.lista = Funciona
12 controlador.lista = Diretor(
13 controlador.lista = Gerente(salario=1000)
14 seguranca = Seguranca(salario=1000)
15 controlador.lista = seguranca
16
17 print 'Bônus total:', controlador.total
18 controlador.excluir(seguranca)
19 print 'Bônus total:', controlador.total
20
```

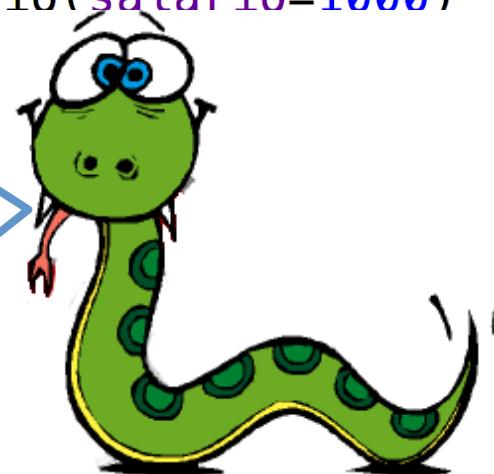


Exercício 15: Teste controle

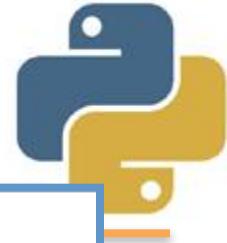


```
1 # -*- coding: UTF-8 -*-
2 # teste_controlador_bonus.py
3
4 from model import Funcionario
5 from model import Seguranca
6 from model import Diretor
7 from model import Gerente
8 from control import ControleBonus
9
10 controlador = ControleBonus()
11 funcionario = Funcionario(salario=1000)
12
13 Importação das classes
14 entidade e da classe de
15 controle
16
17 print 'Bonus total:', contro
18 controlador.excluir(seguranc
19 print 'Bônus total:', contro
20
```

Importação das classes
entidade e da classe de
controle

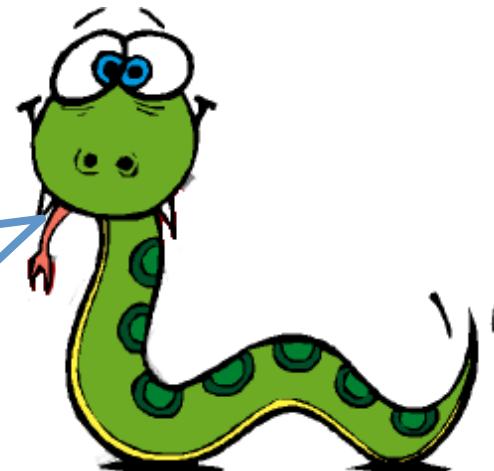


Exercício 15: Teste controle

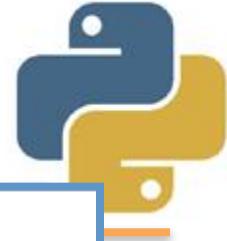


```
1  # -*- coding: UTF-8 -*-
2  # teste_controlador_bonus.py
3
4  from controlador import ControladorBonus
5  from funcionario import Funcionario
6  from diretor import Diretor
7  from gerente import Gerente
8  from seguranca import Seguranca
9
10 controlador = ControladorBonus()
11 controlador.lista = Funcionario(salario=1000)
12 controlador.lista = Diretor(salario=1000)
13 controlador.lista = Gerente(salario=1000)
14 seguranca = Seguranca(salario=1000)
15 controlador.lista = seguranca
16
17 print 'Bônus total:', controlador.total
18 controlador.excluir(seguranca)
19 print 'Bônus total:', controlador.total
20
```

Criação do objeto
controlador

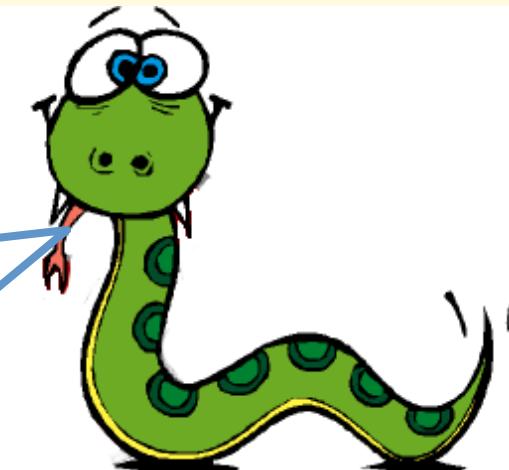


Exercício 15: Teste controle

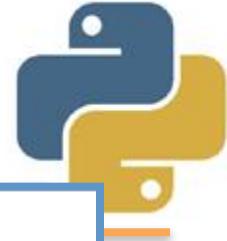


```
1 # -*- coding: UTF-8 -*-
2 # teste_controlador_bonus.py
3
4 from model import Funcionario
5 from model import Seguranca
6
7
8
9
10 controlador = ControleBonus(),
11 controlador.lista = Funcionario(salario=1000)
12 controlador.lista = Diretor(salario=1000)
13 controlador.lista = Gerente(salario=1000)
14 seguranca = Seguranca(salario=1000)
15 controlador.lista = seguranca
16
17 print 'Bônus total:', controlador.total
18 controlador.excluir(seguranca)
19 print 'Bônus total:', controlador.total
20
```

Criação de objetos e
atribuição – setter

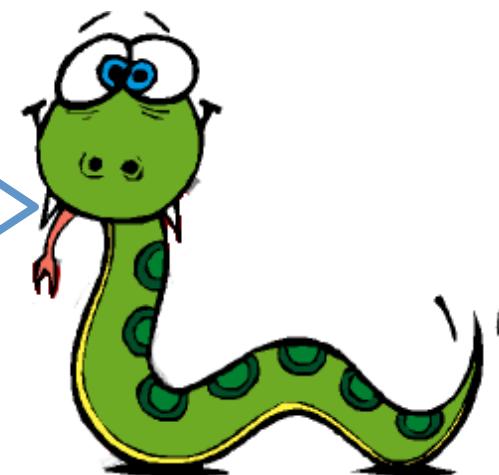


Exercício 15: Teste controle

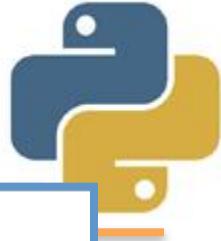


```
1  # -*- coding: UTF-8 -*-
2  # teste_controlador_bonus.py
3
4  from model import Funcionario
5  from model import Seguranca
6  from model import Diretor
7
8
9
10
11
12
13  controlador.lista = Gerente(salario=1000,
14      segurança = Seguranca(salario=1000)
15  controlador.lista = segurança
16
17  print 'Bônus total:', controlador.total
18  controlador.excluir(segurança)
19  print 'Bônus total:', controlador.total
20
```

Outra estratégia para
criação de **objetos** e
atribuição - setter

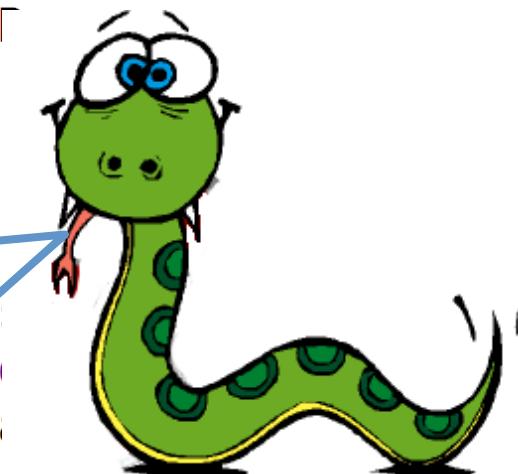


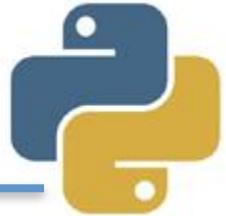
Exercício 15: Teste controle



```
1  # -*- coding: UTF-8 -*-
2  # teste_controlador_bonus.py
3
4  from model import Funcionario
5  from model import Seguranca
6  from model import Diretor
7  from model import Gerente
8  from control import ControleBonus
9
10 controlador = ControleBonus()
11 controlador.lista = Funcionario()
12
13 print 'Bônus total:', controlador.total
14 controlador.excluir(seguranca)
15 print 'Bônus total:', controlador.total
16
17
18
19
20
```

Impressão, exclusão
e nova impressão





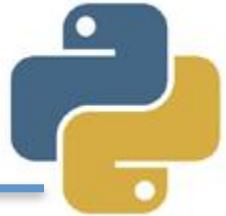
Resultado no console

Run teste_controlador_bonus

```
/Library/Frameworks/Python.framework/Ve
salario: 1000 bonus: 1200.0
salario: 1000 bonus: 1500.0
salario: 1000 bonus: 1200.0
salario: 1000 bonus: 1200.0
Bônus total: 5100.0
Após a exclusão do seguran a
Bônus total: 5100.0
```

Process finished with exit code 0

Resultado no console



Run teste_controlador_bonus

```
/Library/Frameworks/Py
```

```
salario: 1000 bonus: 100
```

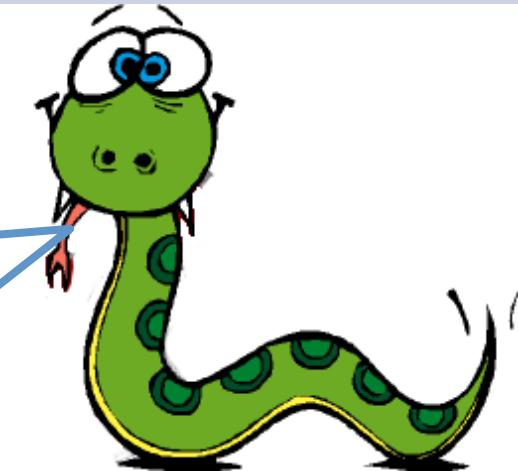
O total de bônus
funcionou para inclusão

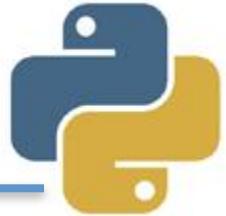
Bônus total: 5100.0

Após a exclusão do segurança

Bônus total: 5100.0

Process finished with exit code 0





Resultado no console

Run teste_controlador_bonus

```
/Library/Frameworks/Python.f
```

```
salario: 1000 bonus: 1200.0
```

```
salario: 1000 bonus: 1500.0
```

```
salario: 1000 bonus: 1500.0
```

```
salario: 1000 bonus: 1500.0
```

```
Bônus
```

Mas o **total** após a
exclusão está **incorrecto**

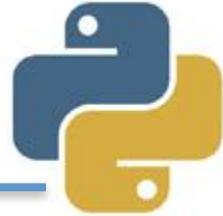
Após a exclusão do segurança

Bônus total: 5100.0



Process finished with exit code 0

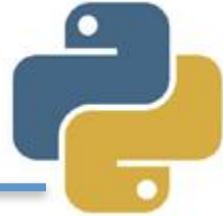
Exercício 16: tirando o bug



- No método **excluir**, **atualize** o total de **bônus**, após **remover** o **item** da lista

```
17     @lista.setter
18     def lista(self, item):
19         #log de funcionários adicionados
20         print 'salario:', item.salario, 'bonus:', item.bonus
21         self.__lista_funcionarios.append(item)
22         self.__total += item.bonus
23
24     def excluir(self, item):
25         self.__lista_funcionarios.remove(item)
26         self.__total -= item.bonus
27
```

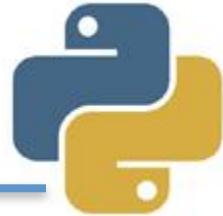
Exercício 16: tirando o bug



- No método **excluir**, **atualize** o total de **bônus**, após **remover** o **item** da lista

```
17     @lista.setter
18     def lista(self, item):
19         #log de funcionários adicionados
20         print 'salario:', item.salario, 'bonus:', item.bonus
21         self.__lista_funcionarios.append(item)
22         self.__total += item.bonus
23
24     def excluir(self, item):
25         self.__lista_funcionarios.remove(item)
26         self.__total -= item.bonus
27
```

Resultado no console

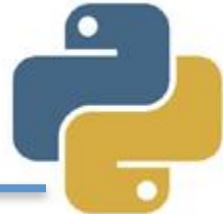


Run teste_controlador_bonus

```
/Library/Frameworks/Python.framework  
salario: 1000 bonus: 1200.0  
salario: 1000 bonus: 1500.0  
salario: 1000 bonus: 1200.0  
salario: 1000 bonus: 1200.0  
Bônus total: 5100.0  
Após a exclusão do segurança  
Bônus soma: 3900.0
```

Process finished with exit code 0

Resultado no console



Run teste_controlador_bonus

/Library/Frameworks/Pyt

salario:

salario:

salario:

salario:

Agora deu

certo

12

15

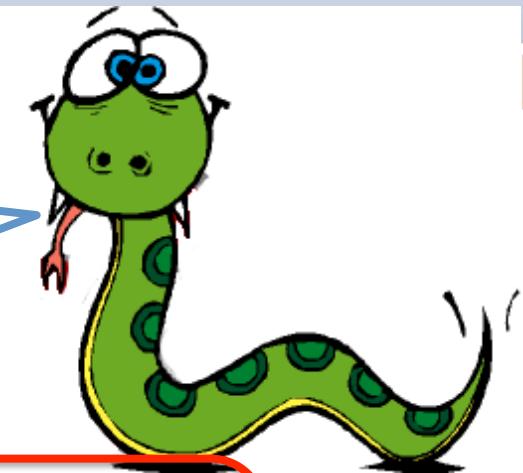
12

12

Bônus total: 5100.0

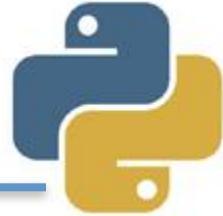
Após a exclusão do segurança

Bônus soma: 3900.0



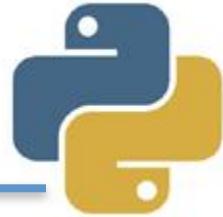
Process finished with exit code 0

Mas a vida...



- Até aqui, nossa estratégia para cálculo do total de bônus depende de dois métodos: **setter** e **excluir**
 - Mas imagine que o salário de um funcionário seja **alterado**, após sua inclusão na lista
 - Nossa estratégia estaria furada 😞
-

Pensamento iterativo

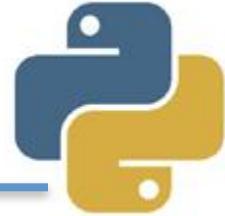


- Poderíamos **criar um método** para percorrer a **lista** e **acumular** o total

```
def soma_iterativa(self):  
    total = 0  
    for item in self.__lista_funcionarios:  
        total += item.bonus  
    return total
```

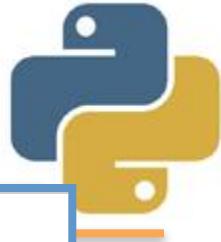
- Isso **resolveria** nosso problema

Pensamento pythônico



- Em python, temos uma **ferramenta** para **transformar** uma **lista** (qualquer iterável) em **outra**: **list comprehensions**
- A função **sum()** sabe somar listas
- Precisamos de uma **lista de bônus**

```
@property
def soma(self):
    lista = [f.bonus for f in self.__lista_funcionarios]
    return sum(lista)
```

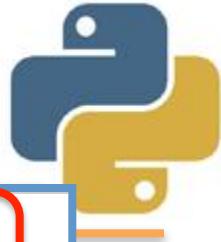


Pensamento pythônico

```
17 print 'Bônus total:', controlador.total
18 print 'Bônus soma: ', controlador.soma
19 controlador.excluir(seguranca)
20 print 'Após a exclusão do segurança'
21 print 'Bônus total:', controlador.total
22 print 'Bônus soma: ', controlador.soma
```

Run teste_controlador_bonus

```
/Library/Frameworks/Python.framework/Versions/2.7/bin/pythonw /Users/luiz/PycharmProjects/teste/teste_controlador_bonus.py
salario: 1000 bonus: 1200.0
salario: 1000 bonus: 1500.0
salario: 1000 bonus: 1200.0
salario: 1000 bonus: 1200.0
Bônus total: 5100.0
Bônus soma: 5100.0
Após a exclusão do segurança
Bônus total: 3900.0
Bônus soma: 3900.0
```

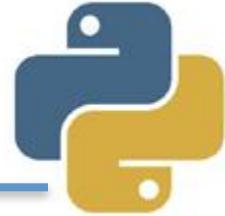


Pensamento pythônico

```
17 print 'Bônus total:', controlador.total
18 print 'Bônus soma: ', controlador.soma
19 controlador.excluir(seguranca)
20 print 'Após a exclusão do segurança'
21 print 'Bônus total:', controlador.total
22 print 'Bônus soma: ', controlador.soma
```

```
Run teste_controlador_bonus
/ Library/Frameworks/Python.framework/Versio
salario: 1000 bonus: 1200.0
salario: 1000 bonus: 1500.0
salario: 1000 bonus: 1200.0
salario: 1000 bonus: 1200.0
Bônus total: 5100.0
Bônus soma: 5100.0
Após a exclusão do segurança
Bônus total: 3900.0
Bônus soma: 3900.0
```

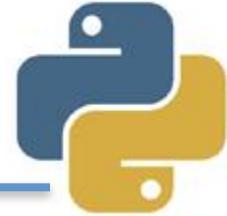
OO e o Polimorfismo



- **Polimorfismo** é a capacidade de tratar instâncias de **classes diferentes** da **mesma forma**. No código a seguir, o **setter** da lista aceita **instâncias de classes distintas**

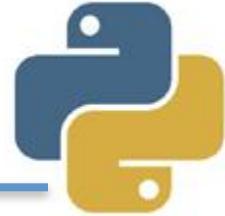
```
11 controlador.lista = Funcionario(salario=1000)
12 controlador.lista = Diretor(salario=1000)
13 controlador.lista = Gerente(salario=1000)
14 segurança = Seguranca(salario=1000)
15 controlador.lista = segurança
```

Polimorfismo pythônico

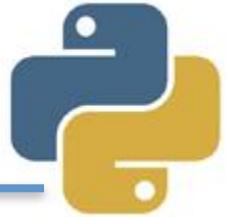


- Em Java e C++, o polimorfismo está fortemente ligado à herança entre classes
 - Mas o python segue o conceito duck typing:
 - Não precisamos saber a classe para invocar um método de um objeto
 - Se o método for definido no objeto, nós podemos invocá-lo
-

Polimorfismo pythônico



- Duck typing foi inspirada no Duck test, atribuído a James Whitcomb Riley:
 - “If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck”
 - “Se uma coisa parece um pato, nada como um pato e fala como pato, então ela provavelmente é um pato”
-

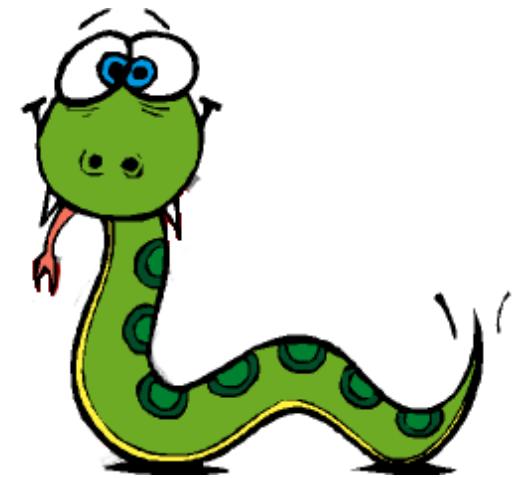


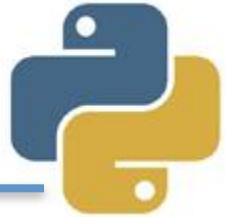
Duck typing

In [2]:

```
class Pessoa:  
    def falar(self):  
        print "Bom dia!"  
  
class Pato:  
    def falar(self):  
        print "Quack quack"  
  
class Cachorro:  
    def falar(self):  
        print "Au au"  
  
class Gato:  
    def falar(self):  
        print "Miau"  
  
def funcao(ser_vivo):  
    ser_vivo.falar()  
  
funcao(Pessoa())  
funcao(Pato())  
funcao(Cachorro())  
funcao(Gato())
```

Bom dia!
Quack quack
Au au
Miau





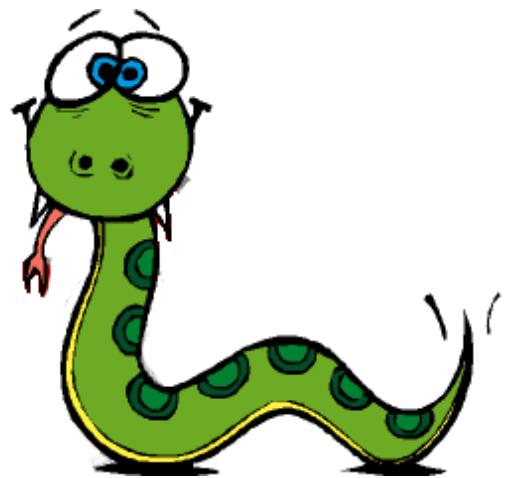
Duck typing

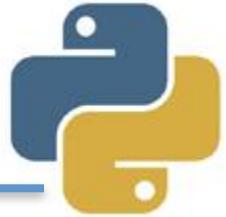
In [2]:

```
class Pessoa:  
    def falar(self):  
        print "Bom dia!"  
  
class Pato:  
    def falar(self):  
        print "Quack quack"  
  
class Cachorro:  
    def falar(self):  
        print "Au au"  
  
class Gato:  
    def falar(self):  
        print "Miau"  
  
def funcao(ser_vivo):  
    ser_vivo.falar()  
  
funcao(Pessoa())  
funcao(Pato())  
funcao(Cachorro())  
funcao(Gato())
```

Bom dia!
Quack quack
Au au
Miau

Definição de **classes**
que possuem o
método **falar()**



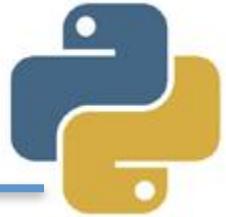


Duck typing

```
In [2]: class Pessoa:  
    def falar(self):  
        print "Bom dia!"  
  
class Pato:  
    def falar(self):  
        print "Quack quack"  
  
class Cachorro:  
    def falar(self):  
        print "Au au"  
  
class Gato:  
    def falar(self):  
        print "Miau"  
  
def funcao(ser_vivo):  
    ser_vivo.falar()  
  
funcao(Pessoa())  
funcao(Pato())  
funcao(Cachorro())  
funcao(Gato())  
  
Bom dia!  
Quack quack  
Au au  
Miau
```

Função que **recebe** um objeto e **invoca** o método **falar()**





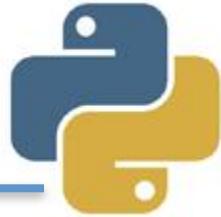
Duck typing

```
In [2]: class Pessoa:  
    def falar(self):  
        print "Bom dia!"  
  
class Pato:  
    def falar(self):  
        print "Quack quack"  
  
class Cachorro:  
    def falar(self):  
        print "Au au"  
  
class Gato:  
    def falar(self):  
        print "Miau"  
  
def funcao(ser_vivo):  
    ser_vivo.falar()  
  
funcao(Pessoa())  
funcao(Pato())  
funcao(Cachorro())  
funcao(Gato())
```

Bom dia!
Quack quack
Au au
Miau

Passagem de referências para a função





Duck typing

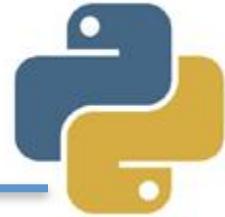
```
In [2]: class Pessoa:  
    def falar(self):  
        print "Bom dia!"  
  
class Pato:  
    def falar(self):  
        print "Quack quack"  
  
class Cachorro:  
    def falar(self):  
        print "Au au"  
  
class Gato:  
    def falar(self):  
        print "Miau"  
  
def funcao(ser_vivo):  
    ser_vivo.falar()  
  
funcao(Pessoa())  
funcao(Pato())  
funcao(Cachorro())  
funcao(Gato())
```

Bom dia!
Quack quack
Au au
Miau

Resultado do
processamento
no notebook



Outro exemplo Duck typing



```
In [4]: def calcular(a, b, c):  
    return (a+b)*c
```

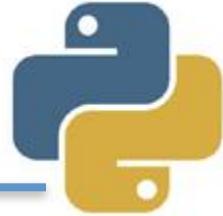
```
print calcular(1, 2, 3)  
print calcular([1, 2, 3], [4, 5, 6], 2)  
print calcular ('maçãs ', 'e laranjas, ', 3)
```

9

[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]

maçãs e laranjas, maçãs e laranjas, maçãs e laranjas,

Outro exemplo Duck typing



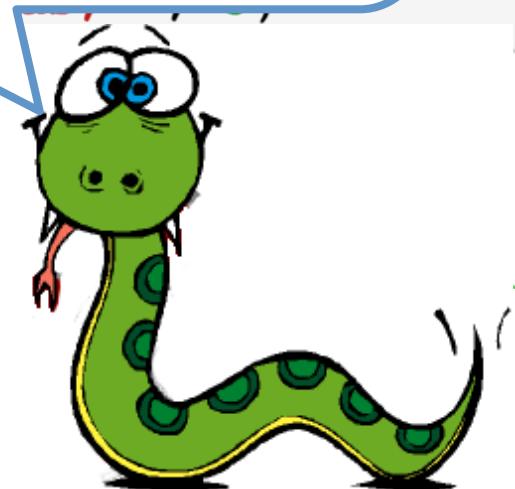
```
In [4]: def calcular(a, b, c):  
    return (a+b)*c
```

```
print calcular(1, 2, 3)  
print calcular([1, 2, 3],  
print calcular ('maçãs ',
```

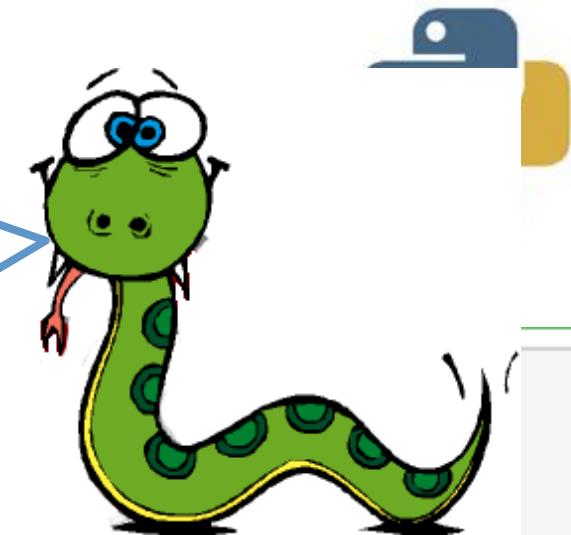
Método que
soma e **multiplica**
três objetos

```
9
```

```
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5,  
maçãs e laranjas, maçãs e laranjas
```



Outro exemplo Duck



Passando três
inteiros

In [4]: `def calcu
return`

```
print calcular(1, 2, 3)
print calcular([1, 2, 3], [4, 5, 6], 2)
print calcular ('maçãs ', 'e laranjas, ', 3)
```

9

[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]

maçãs e laranjas, maçãs e laranjas, maçãs e laranjas,

Outro exemplo Duck



Passando duas
listas e um inteiro

```
In [4]: def calcular(  
    a, b, c):  
    return a + b + c
```

```
print calcular(1, 2, 3)  
print calcular([1, 2, 3], [4, 5, 6], 2)  
print calcular('maças', 'e laranjas', 3)
```

9

```
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
```

maças e laranjas, maças e laranjas, maçãs e laranjas,

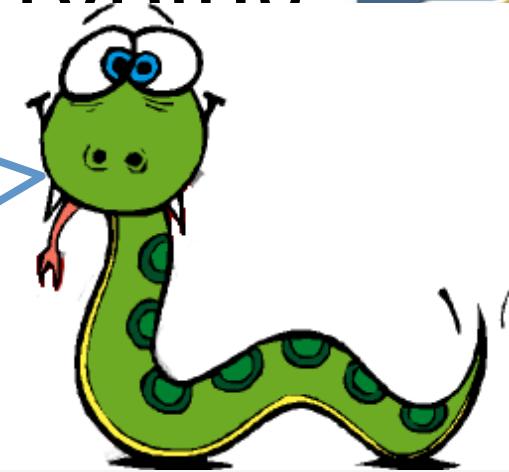
Outro exemplo Duck typing



Passando duas
Strings e um
inteiro

```
In [4]: def calcular(x, y, z):
    return x * y + z

print calcular(1, 2, 3)
print calcular([1, 2, 3], [4, 5, 6], 2)
print calcular ('maçãs ', 'e laranjas, ', 3)
```

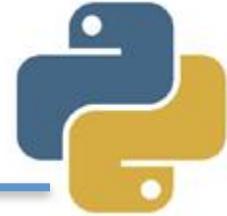


```
9
```

```
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
```

```
maçãs e laranjas, maçãs e laranjas, maçãs e laranjas,
```

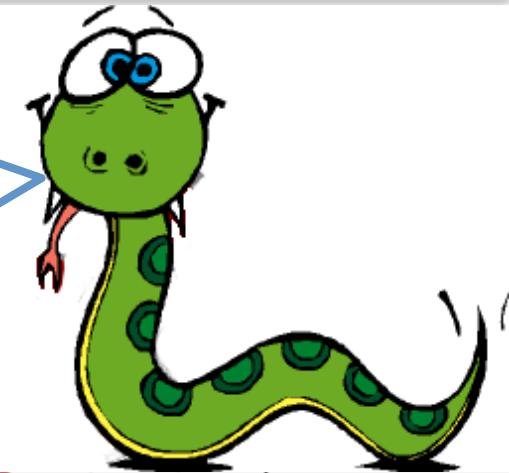
Outro exemplo Duck typing



In [4]:

```
def calculate_fruit_salad(ingredients):
    return ingredients[0] + 6 * ingredients[1]
```

Resultado no
notebook



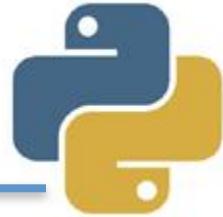
```
9
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
maçãs e laranjas, maçãs e laranjas, maçãs e laranjas,
```



FIM

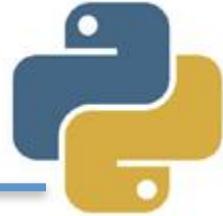


Contatos



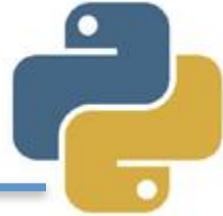
- Márcio Palheta
 - marcio.palheta@gmail.com
 - @marciopalheta
 - <https://sites.google.com/site/marciopalheta/>
-

Bibliografia

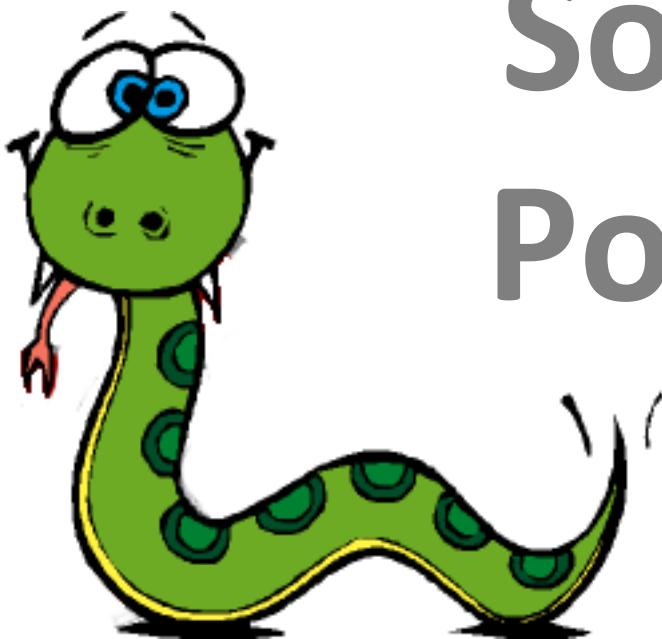


- LIVRO: Apress - Beginning Python From Novice to Professional
 - LIVRO: O'Reilly - Learning Python
 - <http://www.python.org>
 - <http://www.python.org.br>
 - Mais exercícios:
 - <http://wiki.python.org.br/ListaDeExercicios>
 - Documentação do python:
 - <https://docs.python.org/2/>
-

Capítulo 05



Herança, Sobrescrita e Polimorfismo



Márcio Palheta, M.Sc.
marcio.palheta@gmail.com