



Programação Orientada a Objetos

APRESENTAÇÃO DA DISCIPLINA

PROFESSOR:

Demetrios Coutinho

Demetrios.coutinho@ifrn.edu.br

AGENDA

- Herança
- Polimorfismo

OS QUATRO PILARES

ABASTRAÇÃO

ENCAPSULAMENTO

HERANÇA

POLIMORFISMO

Herança

- Tecnicamente, todas as classes que usamos já usam herança, uma vez que elas herdam da classe *object*;
- Os comportamentos fornecidos por essa classe são aqueles com underscore duplo;
- A classe *object* permite que o python trate todos os objetos da mesma maneira, isto é, como um objeto;
- Até agora ao criar nossas classes não explicitamos que herdamos de *object*, mas podemos, conforme segue:

```
1 ▶ class MySubClass(object):  
2     pass
```

Herança

```
1 ▼ class MySubClass(object):  
2     pass
```

- Isso é **herança**;
- Chamamos de superclasse ou classe pai a classe que estendemos e de subclasse ou classe filha a classe que herda.
 - No caso acima, object é a superclasse e MySubClasse a subclasse;
- Para realizar a herança em python basta colocar o nome da superclasse a qual se quer herdar dentro dos parênteses da definição da subclasse;

Herança

- O uso mais simples da herança é adicionar funcionalidade a uma classe existente.
- Para ilustrar, veja um simples exemplo de uma classe base que gerencia contatos:

```
1 ▼ class Contact:  
2     all_contacts = []  
3  
4 ▼     def __init__(self, name, email):  
5         self.name = name  
6         self.email = email  
7         Contact.all_contacts.append(self)
```

- Neste exemplo usamos o que chamamos de variável de classe;
 - A lista `all_contacts` é uma variável de classe porque é compartilhada por todos objetos da classe `Contact`.
- Agora imagine que existem contatos de fornecedores que fazemos pedidos;
 - Poderíamos pensar em adicionar um método `order` na classe `Contact`, mas isso faria que contatos familiares também tivessem esse método;
 - Vamos criar uma classe Fornecedor (*Supplier* em inglês) que além de herdar de `Contact`, também possui o método `order`:

Herança

```
1 v class Supplier(Contact):
2 v     def order(self, pedido):
3 v         print("Pedido para '{}': "
4 v             "'{}'".format(self.name, pedido))
```

```
1 c1 = Contact('Filipe', 'filipe@gmail.com')
2 f1 = Supplier('Amazon', 'amazon@gmail.com')
```

- Note que o objeto *f1* possui **nome** e **email** pois herda da classe **Contact**:

```
1 print(c1.name, c1.email)
2 print(f1.name, f1.email)
```

Filipe filipe@gmail.com
Amazon amazon@gmail.com

Herança

- Perceba ainda que apenas o objeto *f1* é capaz de efetuar pedidos:

```
1 f1.order('Preciso de 2 camisas do palmeiras')
```

Pedido para 'Amazon': 'Preciso de 2 camisas do palmeiras'

```
1 c1.order('Preciso de 2 camisas do palmeiras')
```

```
-----
-----
AttributeError                         Traceback (most recent call
last)
<ipython-input-7-c139743b71d5> in <module>
----> 1 c1.order('Preciso de 2 camisas do palmeiras')

AttributeError: 'Contact' object has no attribute 'order'
```

Herança

- Agora perceba como o atributo `all_contacts` é compartilhado por ambos objetos:

In [8]:

```
1 c1.all_contacts
```

Out[8]:

```
[<__main__.Contact at 0x5018e48>, <__main__.Supplier at 0x5018e10>]
```

In [9]:

```
1 f1.all_contacts
```

Out[9]:

```
[<__main__.Contact at 0x5018e48>, <__main__.Supplier at 0x5018e10>]
```

- Como ele é um atributo de classe, você pode acessá-lo diretamente pela classe:

In [10]:

```
1 Contact.all_contacts
```

Out[10]:

```
[<__main__.Contact at 0x5018e48>, <__main__.Supplier at 0x5018e10>]
```

Herdando de built-in

- Um uso interessante desse tipo de herança é adicionar funcionalidade aos recursos built-in do python.
- Para exemplo, podemos criar uma classe *ContactList* que adiciona um método para fazer busca de contatos em uma lista padrão do python.

```
1  class ContactList(list):
2      def search(self, name):
3          matching_contacts = []
4          for contact in self:
5              if name in contact.name:
6                  matching_contacts.append(contact)
7          return matching_contacts
8
9  class Contact():
10     all_contacts = ContactList()
11
12    def __init__(self, name, email):
13        self.name = name
14        self.email = email
15        Contact.all_contacts.append(self)
```

Herdando de built-in

- Testando:

```
1 c1 = Contact('Fulano da Silva', 'fulano1@dominio.com')
2 c2 = Contact('Fulano Sousa', 'fulano2@dominio.com')
3 c3 = Contact('Ciclano Pereira', 'ciclano3@dominio.com')
```

```
1 Contact.all_contacts.search('Fulano')
```

Out[13]:

```
[<__main__.Contact at 0x50b2be0>, <__main__.Contact at 0x50b2b38>]
```

O que significa essa saída? Por que não aparece o nome e o e-mail?

Sobrescrevendo métodos

- Já falamos da sobrecarga de métodos como `__add__` e `__str__`, basicamente estamos sobrescrevendo a herança dada pela classe `Object`.
- O próprio construtor `__init__` é uma sobrecarga de método.
- Por exemplo, imagine que queremos adicionar o atributo `phone_number` nos contatos dos nossos amigos mais próximos.

```
1 ▼ class Friend(Contact):
2 ▼     def __init__(self, name, email, phone):
3         self.name = name
4         self.email = email
5         self.phone = phone
```

- Observe 2 problemas com o código acima:
 1. Repetimos código o que pode tornar o processo de manutenção do sistema ruim;
 2. Esquecemos de adicionar o contato do amigo na variável de classe `all_contacts`;

Sobrescrevendo métodos

- O que nós realmente precisamos é executar o `__init__` da classe *Contact* e apenas adicionar o telefone do contato.
- Para tanto, podemos usar a palavra chave **super**, que nos permite invocar métodos da **superclasse** diretamente:

```
1 class Friend(Contact):
2     def __init__(self, name, email, phone):
3         super().__init__(name, email)
4         self.phone = phone
```

```
1 friend1 = Friend('Beltrano Ferreira', 'beltrano@dominio.com', '2121-3131')
```

```
1 friend1.phone
```

Qualquer método, atributo **não-privado** é herdado pela superclasse.

Sobrescrevendo métodos

- Mais exemplos:

```
class Funcionario:
```

```
def __init__(self, nome, cpf, salario):  
    self._nome = nome  
    self._cpf = cpf  
    self._salario = salario
```

```
# outros métodos e properties
```

```
def get_bonificacao(self):  
    return self._salario * 0.10
```

```
class Gerente(Funcionario):
```

```
def __init__(self, nome, cpf, salario, senha, qtd_gerenciaveis):  
    super().__init__(nome, cpf, salario)  
    self._senha = senha  
    self._qtd_gerenciaveis = qtd_gerenciaveis
```

```
def get_bonificacao():
```

```
    return super().get_bonificacao() + 1000
```

```
# métodos e properties
```

Sobrescrevendo métodos

- Mais exemplos:

```
funcionario = Funcionario('João', '1111111111-11', 2000.0)
print(vars(funcionario))
```

```
gerente = Gerente('José', '222222222-22', 5000.0, '1234', 0)
print(vars(gerente))
```

Saída:

```
{'_salario': 2000.0, '_nome': 'João', '_cpf': '1111111111-11'}
{'_cpf': '222222222-22', '_salario': 5000.0, '_nome': 'José', '_qtd_funcionarios': 0, '_senha': '1234'}
```

Polimorfismo

- **Polimorfismo** é a capacidade do objeto de uma subclasse ser referenciado como uma superclasse para que, dependendo da subclasse, possa haver comportamentos diferentes.
- Em outras palavras, diferentes comportamentos podem ocorrer dependendo de qual subclasse está sendo usada, onde não existe a necessidade de sabermos explicitamente que subclasse é essa.

Polimorfismo

- Vamos ver um exemplo.

Criando um jogo de RPG

ATRIBUTOS DOS PERSONAGENS

- Raça
- Atributos
 - Força
 - Destreza
 - Constituição
 - Sabedoria
 - Inteligência
 - Carisma

Atributos

FORÇA

FORÇA

QUANTIFICA O PESO QUE PODE SER CARREGADO
E O PODER DE SEUS ATAQUES FÍSICOS



INCORPÓREO (0)
NÃO POSSUI FORÇAS PARA GERAR ENERGIA E MOVER MATÉRIA

INCAPAZ (1-5)
NECESSITA DE MUITO EFORÇO PARA REALIZAR TAREFAS SIMPLES



FRACO (6-9)
POSSUI DIFICULDADE EM LEVANTAR CARGAS MAiores E REALIZAR EXERCÍCIOS FÍSICOS



MEDIANO (10-11)
CONSEGUE CARREGAR UM CERTO PESO E VENCER ALGUMAS LUTAS



[FACEBOOK/ATAVERNA](#)

FORÇA

QUANTIFICA O PESO QUE PODE SER CARREGADO
E O PODER DE SEUS ATAQUES FÍSICOS



FORTE (12-15)
INFLIGE GOLPES MAIS PODEROSOS E CARREGA MAIS PESO QUE O NORMAL



SUPER PODEROSENDO (16-20)
POSSUI UMA FORÇA ALÉM DO COMUM E COM EFORÇO PODE CARREGAR PESOS ABSURDOS



IMBATÍVEL (21+)
PODE CARREGAR CARGAS ENORMES SEM EFORÇO E POSSUI UMA FORÇA IMENSURÁVEL



[FACEBOOK/ATAVERNA](#)

Atributos

DESTREZA

DESTREZA

QUANTIFICA SUAS HABILIDADES DE PRECISÃO
ASSIM COMO SUA AGILIDADE E EQUILÍBRIO



DESACORDADO (0)
NÃO PODE SE MOVER
DE MANEIRA ALGUMA



ABATIDO (1-5)
CONSEGUE REALIZAR POUcos
MOVIMENTOS MUITO LENTAMENTE



DESAJEITADO (6-9)
POSSUI POUCA HABILIDADE MOTORA E REALIZA ALGUNS
MOVIMENTOS SIMPLES DE FORMA ATRAPALHADA



REGULAR (10-11)
REALIZA MOVIMENTOS SIMPLES DE FORMA
ACEITÁVEL MAS NÃO POSSUI MUITA AGILIDADE



FACEBOOK/ATAVERNA

DESTREZA

QUANTIFICA SUAS HABILIDADES DE PRECISÃO
ASSIM COMO SUA AGILIDADE E EQUILÍBRIO



ÁGIL (12-15)
CONSEGUE REALIZAR MOVIMENTOS RÁPIDOS
E ÁGEIS COM BOA PRECISÃO



NINJA (16-20)
POSSUI UMA AGILIDADE INCRÍVEL E
UMA PRECISÃO FORA DO NORMAL



IMPERCEPTÍVEL (21+)
É CAPAZ DE REALIZAR MOVIMENTOS TÃO
PERFEITOS E RÁPIDOS QUE PASSAM
DESPERCEBIDOS PELA MAIORIA



FACEBOOK/ATAVERNA

Atributos

CONSTITUIÇÃO

CONSTITUIÇÃO

QUANTIFICA SUA RESISTÊNCIA CORPORAL TANTO DE FORMA EXTERNA COMO INTERNA



ESPECTRO (0)
NÃO POSSUI UM CORPO FÍSICO E MATERIAL

ENFERMO (1-5)
POSSUI UMA SAÚDE CRITICAMENTE COMPROMETIDA



FRÁGIL (6-9)
DESMAI A E CANS A FACILMENTE POR TER UM CORPO FRACO



SAUDÁVEL (10-11)
FICA FATIGADO DEPOIS DE UM CERTO TEMPO DE CORRIDA E É SUSCETÍVEL À NOCAUTES



FACEBOOK/ATAVERNA

CONSTITUIÇÃO

QUANTIFICA SUA RESISTÊNCIA CORPORAL TANTO DE FORMA EXTERNA COMO INTERNA



DURÃO (12-15)
AGUENTA MAIS IMPACTO DO QUE A MAIORIA E NÃO CANS A FACILMENTE



SUPER RESISTÊNTE (16-20)
POSSUI UMA RESISTÊNCIA FÍSICA INCOMUM E QUASE NUNCA FICA CANSADO



IMORTAL (21+)
NUNCA FICA CANSADO E CONSEGUE RESISTIR AOS MAIS FORTES GOLPES E VENENOS



FACEBOOK/ATAVERNA

Atributos

SABEDORIA

SABEDORIA

QUANTIFICA A SAGACIDADE EM UTILIZAR MÉTODOS E CONHECIMENTOS DE SENSO-COMUM



INCONSCIENTE (0)
NÃO POSSUI A CAPACIDADE DE RACIOCÍNIO



IRRACIONAL (1-5)
AGE SEM RAZÃO E PARECE NÃO NOTAR COISAS IMPORTANTES À SUA VOLTA



DESATENTO (6-9)
OUASE NÃO POSSUI ALGUM SENSO COMUM E AGE POR IMPULSO



SIMPLES (10-11)
É CAPAZ DE PLANEJAR COISAS SIMPLES E POSSUI ALGUM SENSO COMUM



FACEBOOK/ATAVERNA

SABEDORIA

QUANTIFICA A SAGACIDADE EM UTILIZAR MÉTODOS E CONHECIMENTOS DE SENSO-COMUM



PERSPICAZ (12-15)
CONSEGUE PERCERER QUANDO ALGO PARECE ESTRANHO APENAS USANDO A INTUIÇÃO



FILÓSOFO (16-20)
POSSUI A CAPACIDADE DE PERCERER SITUAÇÕES FACILMENTE E É CONSTANTEMENTE PROCURADO COMO CONSELHEIRO



ILUMINADO (21+)
CARREGA UMA INTUIÇÃO INCRÍVEL E É CAPAZ DE PREDIZER FUTURAS SITUAÇÕES COM RICOS DETALHES



FACEBOOK/ATAVERNA

Atributos

INTELIGÊNCIA

INTELIGÊNCIA

QUANTIFICA A RACIONALIDADE E A BAGAGEM INTELECTUAL DE CONHECIMENTOS CIENTÍFICOS



INANIMADO (0)
NÃO É CAPAZ DE ADQUIRIR CONHECIMENTO E RACIOCINAR



INCIVILIZADO (1-5)
AGE APENAS POR INSTINTO COMO UM ANIMAL SELVAGEM



PARVO (6-9)
APRENDE MAIS LENTAMENTE QUE O NORMAL E DEMORA PARA RACIOCINAR



MEDÍOCRE (10-11)
É CAPAZ DE APRENDER NORMALMENTE E USAR A LÓGICA EM ALGUMAS SITUAÇÕES



FACEBOOK/ATAVERNA

INTELIGÊNCIA

QUANTIFICA A RACIONALIDADE E A BAGAGEM INTELECTUAL DE CONHECIMENTOS CIENTÍFICOS



ESTUDADO (12-15)
POSSUI UMA FACILIDADE MAIOR PARA APRENDER E CARREGA UMA BAGAGEM DE CONHECIMENTO



GÊNIO (16-20)
SOLUCIONA PROBLEMAS DE LÓGICA RAPIDAMENTE E POSSUI UM GRAU DE CONHECIMENTO MUITO ELEVADO



$$X_t = \sum_{i=1}^{n_t} (X_{t,i} - \bar{X}_t)^2$$

ONISCIENTE (21+)
CARREGA UMA MENTE EXTRAORDINÁRIA QUE POSSUI UMA QUANTIDADE INCRÍVEL DE CONHECIMENTO E UMA VELOCIDADE DE PENSAMENTO ABSURDA



FACEBOOK/ATAVERNA

Atributos

CARISMA

CARISMA

QUANTIFICA A CAPACIDADE DE SOCIALIZAR E DE DOBRAR OUTROS SERES ATRAVÉS DE EXPRESSÕES



ABERRAÇÃO (0)

NÃO É CAPAZ DE SE EXPRESSAR E AFASTA OUTRAS PESSOAS



INEXPRESSIVO (1-5)

POSSUI UMA EXTREMA FALTA DE EMPATIA E RARAMENTE DEMONSTRA EXPRESSÕES



RUDE (6-9)

É POUCO SOCIALÉVEL E QUASE NUNCA CONSEGUE SE COMUNICAR DE FORMA CORRETA



SOCIÁVEL (10-11)

CONSEGUE CONVERSAR NORMALMENTE COM OUTRA PESSOA DE FORMA AMIGÁVEL

[FACEBOOK/ATAVERNA](#)



CARISMA

QUANTIFICA A CAPACIDADE DE SOCIALIZAR E DE DOBRAR OUTROS SERES ATRAVÉS DE EXPRESSÕES



PERSUASIVO (12-15)

POSSUI A HABILIDADE DE CONVENCER OU INTIMIDAR OUTRAS PESSOAS FACILMENTE



INFLUENTE (16-20)

É EXTREMAMENTE CONHECIDO E INFLUENTE CONSEGUINDO QUASE TUDO O QUE QUISER



ÍDOLO (21+)

SEMPRE CONSEGUE O QUE DESEA E É TRATADO QUASE COMO UM DEUS PELAS PESSOAS



[FACEBOOK/ATAVERNA](#)

Raças

RAÇAS E SUB-RAÇAS

Dwarf (Anão) +2 CON

– Hill Dwarf (Anão da Colina) +1 SAB

– Mountain Dwarf (Anão da Montanha) +2 FOR



Elf (Elfo) +2 DES

– High Elf (Alto Elfo) +1 INT

– Wood Elf (Elfo da Floresta) +1 SAB

– Dark Elf 'Drow' (Elfo Negro 'Drow') +1 CAR



Barbaro +2 DES

– Lightfoot (Pés Leves) +1 CAR

– Stout (Robusto) +1 CON

Raças

RAÇAS E SUB-RAÇAS

Human (Humano) +1 p/ todos os atributos



Dragonborn (Draconato) +2 FOR e +1 CAR



Gnome (Gnomo) +2 INT

- Forest Gnome (Gnomo da Floresta) +1 DES
- Rock Gnome (Gnomo da Pedra) +1 CON



Tiefling (Ladrão) +1 INT e +2 CAR



DÚVIDAS?

