


DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Orientação a Objetos



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

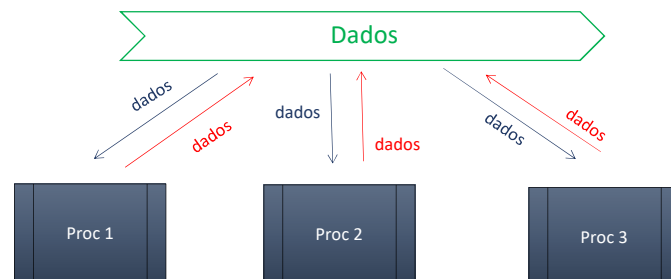
Orientação a Objetos

Python é uma linguagem orientada a objetos.

O que significa?

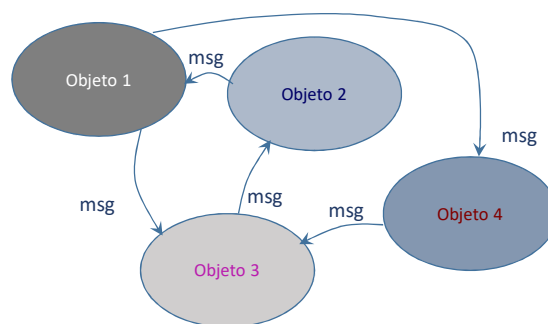
2

Orientação a Procedimentos




Os dados estão associados a variáveis e são compartilhados pelas funções. Uma função recebe os dados que necessita, opera sobre eles e retorna um resultado que pode modificar o estado original.
Estado inicial → sequência de procedimentos → Estado Final

Orientação a Objetos



Os objetos possuem atributos (valores que definem o objeto) e métodos (ações que o objeto sabe realizar).
Os objetos se comunicam por meio de mensagens.
Um objeto responde à mensagem recebida de acordo com seus métodos, alterando seus atributos e/ou enviando mensagens.



Orientação a Procedimentos

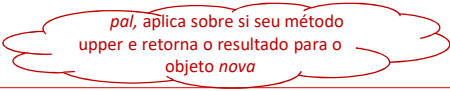
X

Orientação a Objetos

Exibir um texto em caixa alta.

<pre>def Maiuscula(c) : letras='abcdefghijklmnopqrstuvwxyzçáãêéíóôû' if c in letras: return chr(ord(c) - ord('a') + ord('A')) return c def tornaMaiusc(pal) : nova="" for c in pal: nova=nova+Maiuscula(c) return nova pal="transformando todas as letras em maiúsculas!!!" nova=tornaMaiusc(pal) print(nova)</pre>	<div style="border: 1px solid blue; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="writing-mode: vertical-rl; transform: rotate(180deg);">O</div> <div style="writing-mode: vertical-rl; transform: rotate(180deg);">P</div> </div>
---	--


```
pal="transformando todas as letras em maiúsculas!!!"
nova=pal.upper().
print(nova)
```



O

O

5



Orientação a Procedimentos

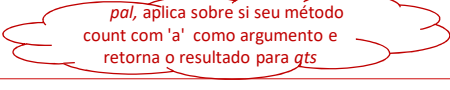
X

Orientação a Objetos

Contar as ocorrências de um certo caractere em um texto.

<pre>def contar(pal,l) : qt=0 for c in pal: if c==l: qt+=1 return qt pal="contando quantos 'a'" qtos=contar(pal,'a') print(qtos)</pre>	<div style="border: 1px solid blue; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="writing-mode: vertical-rl; transform: rotate(180deg);">O</div> <div style="writing-mode: vertical-rl; transform: rotate(180deg);">P</div> </div>
---	--


```
pal="transformando todas as letras em maiúsculas!!!"
qtos=pal.count('a')
print(qtos)
```



O

O

6




Objeto

Objeto possui:

- atributos (o que o objeto tem...)
- comportamento (o que o objeto faz ...)

O comportamento de um objeto é o que o objeto sabe e pode fazer. O comportamento de um objeto é traduzido em métodos (funções internas do objeto).

7



Uma Pessoa como um Objeto

Uma forma simplificada de pensar uma pessoa como um objeto:

O que uma pessoa tem? (Quais são os atributos de uma pessoa?)


- nome
- idade

Qual o comportamento de uma pessoa ? (O que uma pessoa pode fazer?)

Apresentar-se (dizendo qual seu nome e sua idade);

Fazer aniversário;

8



Representando uma Pessoa

Uma lista? Um dicionário? Uma string e um inteiro?

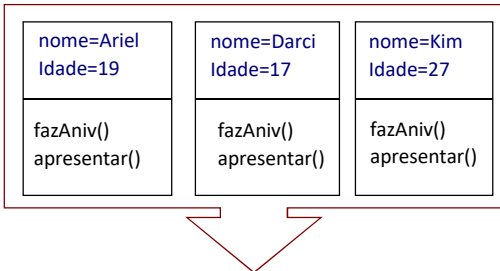
Um novo tipo com dois valores e que realize as operações desejadas?

Criar uma Classe Pessoa: estrutura de uma pessoa + operações
atributos relevantes + comportamento (métodos)

e **instâncias desta classe (objetos):** para associar os dados das pessoas.


Classe

Pessoa	
atributos	nome: idade:
métodos	fazAniv() apresentar()



Objetos

9



Objetos da classe Pessoa

```

classe Pessoa( ):
    def constroi(nome,idade):
        ...
    def exhibe():
        ....
    def fazAniv():
        ....

p1= Pessoa('Darci',17)
p1.exibe()
p2= Pessoa('Kim',27)
p2.exibe()
p3= Pessoa('Ariel',19)
p3.exibe()
p1.fazAniv()
p1.exibe()

```

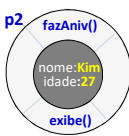
Nome:Darci Idade: 17

Nome:Kim Idade: 27

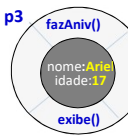
Nome:Ariel Idade: 19

Nome:Darci Idade: 18

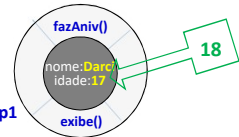
p2




p3



p1




10



Orientação a Objetos em Programação

- Surgiu da necessidade de representar e manipular informações mais complexas.
- É uma forma de entender um problema e desenvolver sua solução.
- Os **elementos conceituais** do programa são representados como **objetos**:
 - *Um objeto é algo que tem significado para a aplicação. Mantém e permite a manipulação dos valores.*
- Um **programa orientado a objetos** é organizado como uma coleção de **objetos** que **interagem** para realizar as tarefas
 - ✓ Cada objeto tem seus valores (atributos)
 - ✓ Cada objeto tem um papel a cumprir
 - ✓ Cada objeto oferece um conjunto de serviços ou realiza um conjunto de ações

11



Orientação a Objetos em Programação

Conceitos envolvidos:

- ✓ Classe
- ✓ Objeto
- ✓ Mensagem

Alguns princípios envolvidos:

- ✓ Abstração
- ✓ Encapsulamento
- ✓ Polimorfismo
- ✓ Composição

12

O que é um Objeto

O objeto é uma representação de qualquer coisa: real ou abstrata.

Os objetos podem ser:

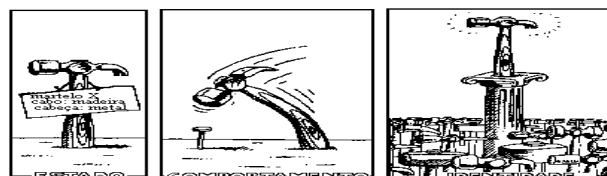
- Físicos: um carro, um relógio, uma pessoa, um cachorro, um pássaro
- Conceitos: uma matriz, uma data, um horário
- Entidade de Software: um arquivo, uma lista, um botão de uma interface, a turtle, uma string, um inteiro, um float

É capaz de reagir a mensagens enviadas a ele, se relacionar e enviar mensagens a outros objetos ou a ele mesmo

13

O que é um Objeto

Entidade com fronteiras bem definidas e que possui identidade, estado e comportamento próprios;



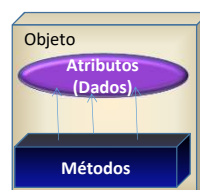
- Identidade - o que diferencia um objeto de outro.
- Comportamento - é definido pelas suas operações (métodos), isto é, como o objeto age e/ou reage às requisições de outros objetos, em termos de mudanças de estados e passagem de mensagens
- Estado - definido pelos valores dos seus atributos

14

Objetos

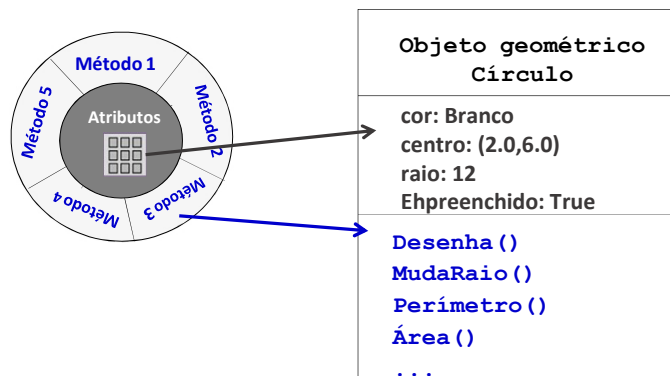
Um objeto é composto por :

- **Atributos:** características ou propriedades que o definem.
Seus valores definem o estado do objeto em um determinado instante
- **Métodos:** conjunto de ações pré-definidas que o objeto pode executar.
Funções e/ou procedimentos contidos no objeto que descrevem seu comportamento.



15

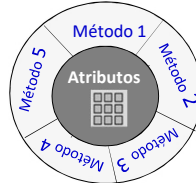
Objetos: Exemplo



16

Objetos: Encapsulamento

Nesse modelo de objeto, os atributos são **mantidos** no seu centro, ou núcleo.



As operações (métodos) rodeiam e escondem o núcleo de um objeto dos demais objetos existentes em um programa ("caixa preta").

O empacotamento da estrutura interna de um objeto é denominado **encapsulamento**.

O encapsulamento de um objeto determina a permissão que outros objetos terão para acessar seus atributos (estado):

- ✓ *A interface declara todas as operações permitidas (métodos).*
- ✓ *O acesso aos dados é feito pela ativação de um método.*
- ✓ *Mudanças internas do objeto não afetam o resto do sistema.*

17


Objetos - Exemplo

Matriz A:

- atributos:
 - linhas,
 - colunas
 - elementos da matriz.
- funcionalidades associadas:
 - inicialização,
 - leitura,
 - multiplicação por um vetor,
 - multiplicação por um escalar,
 - inversão,
 - determinante...

$$A = \begin{bmatrix} 3 & 5 & 1 \\ 2 & -1 & 0 \\ -1 & 3 & 1 \end{bmatrix}$$

18



Mão na Massa

1) Defina um usuário para entrar no ambiente Ead:


Atributos:

- tipo: Aluno ou Professor/Funcionário
- login: matrícula/identificação
- senha

Métodos:

- Autenticar (entrar)

19



O que é uma classe

Uma classe é uma **abstração** das características **relevantes** de um grupo de coisas do mundo real: o molde para a criação do objeto.

Uma classe é a descrição de um conjunto de objetos que possuem a mesma semântica e compartilham as mesmas propriedades (atributos, operações e relacionamentos).

Grady Booch
UML User Guide

20

Abstração

Uma **abstração** é qualquer modelo que inclui os aspectos **relevantes de alguma coisa**, ao mesmo tempo em que ignora os menos importantes.



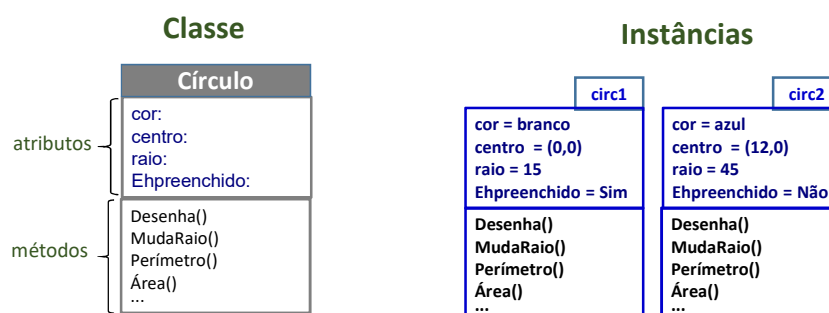
21

O que é uma classe


A classe fornece a estrutura para a construção de objetos.

É um modelo para a criação de objetos

Um objeto é uma instância de uma classe.



22



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Exemplo Classe e Objetos

Classe
Retângulo

dimensões
 cor
 posição
 orientação
 largura
 altura

Área()
 Perímetro()
 TrocarCor()
 TrocarOrientação()
 Desenha()

Objetos

retangulo1

dimensões = 2
 cor = azul
 posição = (10,100)
 orientação = 0 graus
 largura = 50
 altura = 100

Área()
 Perímetro()
 TrocarCor()
 TrocarOrientação()
 Desenha()



retangulo2

dimensões = 2
 cor = rosa
 posição = (10,0)
 orientação = 45graus
 largura = 20
 altura = 35


Área()
 Perímetro()
 TrocarCor()
 TrocarOrientação()
 Desenha()

```
retangulo1.Desenha()
```

```
retangulo2.Desenha()
```

23



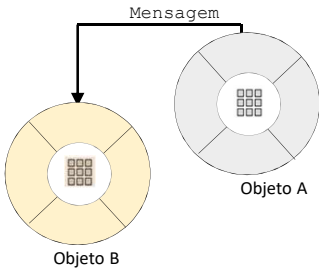
DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Mensagens


Objetos interagem através de **mensagens**;

Quando um objeto **A** deseja que um objeto **B** execute uma de suas operações, **A** envia uma mensagem para **B**;

As informações passadas através de uma mensagem são os **parâmetros** da operação a ser executada.



24

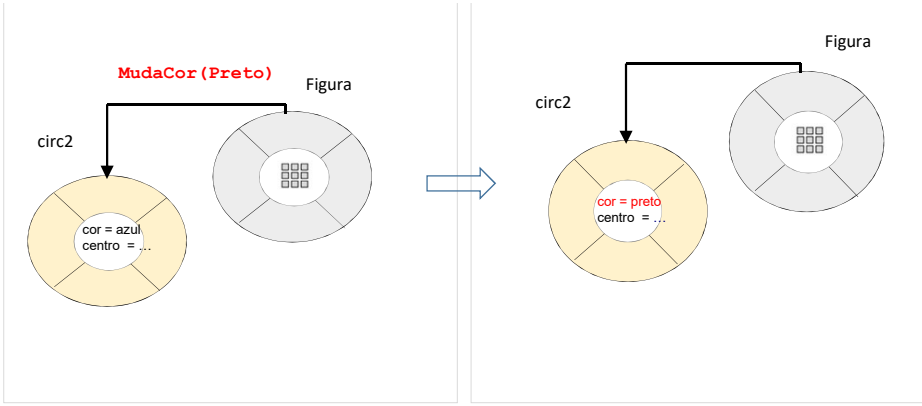


DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO


Mensagens

Os *métodos* são os *veículos de comunicação* entre objetos e operam sobre o *estado interno* de um objeto.

Exemplo 1:




25



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Orientação a Objetos em Python

26



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

OO em Python

Em Python, todos os valores são objetos:

- tartaruga (*Turtle*)


```
>>> pat=turtle.Turtle()
>>> type(pat)
<class 'turtle.Turtle'>
```
- tupla (*tuple*)


```
>>> type((2,3))
<class 'tuple'>
>>> isinstance((2,3),tuple)
True
>>> isinstance((2,3),float)
False
```
- arquivo

```
>>> arq = open('arq.txt','r')
>>> type(arq)
<class '_io.TextIOWrapper'>
```


- cadeia de caracteres (*string*)


```
>>> x='banana'
>>> type(x)
<class 'str'>
>>> isinstance(x,str)
True
```
- lista (*list*)


```
>>> l=[1,2,3,4,5]
>>> type(l)
<class 'list'>
>>> isinstance(l,list)
True
```
- dicionário (*dict*)


```
>>> d={'ana':9,'pedro':8.5}
>>> type(d)
<class 'dict'>
>>> isinstance(d,list)
False
```

27

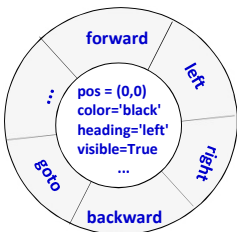


DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

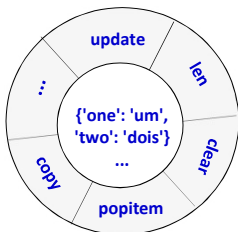
OO em Python

Estado e Métodos

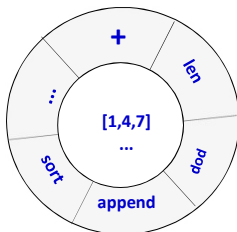
Turtle *pat*



Dicionário *IngPort*



Lista *Lnum*



Lembrando: Cada objeto é uma **instância** de uma classe e possui:

- ✓ um **estado** — valores que caracterizam o próprio objeto e que o distingue dos outros;
- ✓ uma coleção de **métodos** — ações que o objeto pode executar.

28

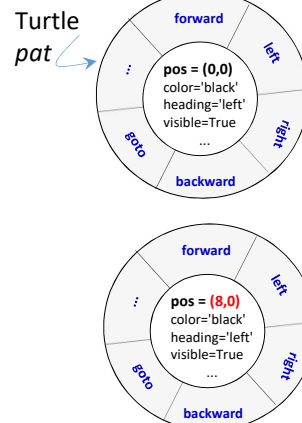
OO em Python

Programas utilizam os objetos em cálculos para computar valores e/ou executando os métodos associados;

```
import turtle

pat=turtle.Turtle()

pat.goto(8,0)
```



29

Definindo novas Classes

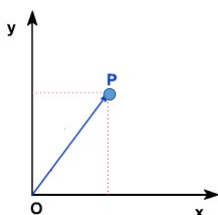
Linguagens orientadas a objetos permitem a definição de novas classes.

- ✓ *As classes criadas pelo programador modelam as estruturas de dados dos elementos conceituais do programa .*
- ✓ *Uma classe cria um novo tipo de dado*

30

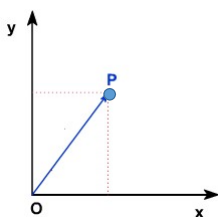
Exemplo

Faça um programa que mostre a distância que um ponto P está da origem. As coordenadas x e y do ponto P são fornecidas pelo usuário.



31

Exemplo



Elemento conceitual manipulado pelo programa:

- Ponto

32



Como representar os pontos no plano?

Um ponto é caracterizado por duas coordenadas numéricas: (x,y)

- ✓ Os dois valores devem ser agrupados em um único objeto

Como representá-los?

- a) Lista
- b) tupla
- c) **novo tipo composto por dois valores** definidos pelo programador



Uma classe

Quais operações devem ser definidas?

- a) consultar/alterar coordenadas
- b) exibir o ponto no formato usual: (x,y)
- c) calcular a distância entre dois pontos
- d) ...

33



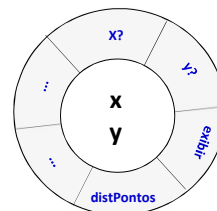
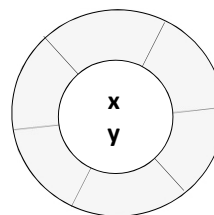
A classe ponto

Atributos:

- coordenada x
- coordenada y

Métodos:

- consultar a coordenada x
- consultar a coordenada y
- alterar a coordenada x
- alterar a coordenada y
- exibir o ponto no formato usual: (x,y)
- calcular a distância do ponto à origem
- calcular a distância entre dois pontos
- desloca n unidades
-



34

Criando classes

```
class NomedaClasse:
    """Texto com a documentação da classe"""
    #corpo da classe
    atr = valor
    ...
    atr = valor
    def metodo (self, ... param):
        ...
    def metodo (self, ... param):
```

- Texto com a documentação da classe: texto explicativo sobre os atributos e métodos da classe
- Corpo da classe: bloco de código indentado que contém os atributos e métodos que pertencem à classe.
- Uma instância da classe é criada usando *NomedaClasse ()*

35

Exemplo

atributos de dados na classe funcionam como valores *default* para os atributos das instâncias

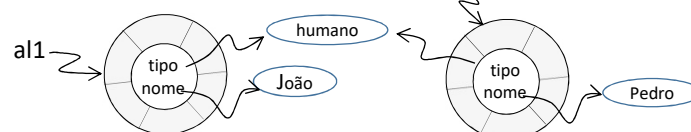
Método construtor de instâncias

```
class Aluno:
    tipo = 'humano' # atributo de classe
    def __init__(self, nome):
        self.nome = nome # atributo de instância
```


atributos da instância só podem ser acessados via self

```
>>> al1 = Aluno('João')
>>> al2 = Aluno('Pedro')
```

cria objeto do tipo Aluno



36



Exemplo

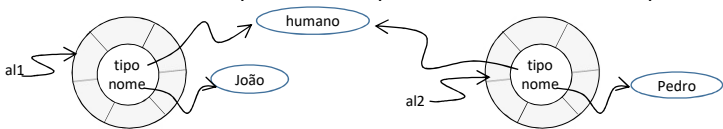
Atributo de classe: definido na classe. São variáveis pré-definidas que estarão presentes nas instâncias desta classe

Atributo de instância: definido no objeto. Dois objetos distintos de uma mesma classe, podem ter conjuntos diferentes de atributos

Todo atributo de classe é refletido nas instâncias;


Atributos de instâncias são únicos e pertencem apenas a uma instância específica;

Acessa
atributo
do objeto



>>> al1.tipo	# copiado pelos objetos do tipo Aluno	'humano'
>>> al2.tipo	# copiado pelos objetos do tipo Aluno	'humano'
>>> al1.nome	# único do objeto al1	'João'
>>> al2.nome	# único do objeto al2	'Pedro'

37



O método especial `__init__`

Construtor da classe.


- Cria e preenche os atributos do objeto na sua instanciação.
- É invocado automaticamente na criação do objeto
- Se não for definido, o Python atribui a essa classe um construtor vazio

```
class NomedaClasse:
    """Texto com a documentação da classe"""
    def __init__(self, params):
        i_1
        i_n
```

self : referência à instância que chamou o método

params: Opcionais. Especificam os parâmetros do método

38




DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Classe Ponto sem parâmetros

```
class Ponto:
    """ classe Ponto: representa e manipula coordenadas x,y """
    def __init__(self):
        """ cria um novo ponto na origem """
        self.x = 0
        self.y = 0
        return
```

Acessa atributo
x do objeto
que ativou o
método

39



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Instanciando dois pontos

```
class Ponto:
    """ classe Ponto:
    representa e manipula coordenadas x,y
    """
    def __init__(self):
        """ cria um ponto na origem """
        self.x = 0
        self.y = 0
        return
```

```
>>> p1=Ponto()
>>> p2=Ponto()
>>> print(p1)
>>> print(p2)
```

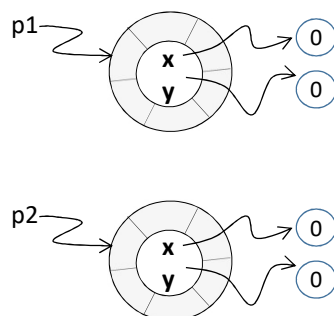


Diagram illustrating the instantiation of two objects, p1 and p2. Each object is represented by a circle containing 'x' and 'y' attributes, both pointing to the value 0.

Como
exibir os
valores?

```
<__main__.Ponto object at 0x1006e5b50 >
<__main__.Ponto object at 0x1006e5b90 >
```

40



Classe Ponto com parâmetros

```
class Ponto:
    """ classe Ponto: representa e manipula coordenadas x,y """
    def __init__(self,x,y):
        """ cria um novo ponto (x,y) """
        self.x = x
        self.y = y
        return
```

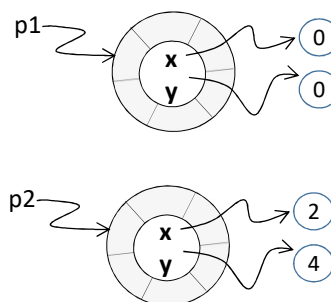
41



Instanciando dois pontos

```
class Ponto:
    """ classe Ponto:
        representa e manipula coordenadas x,y
    """
    def __init__(self,x,y):
        """ cria novo ponto (x,y) """
        self.x = x
        self.y = y
        return
```

```
p1=Ponto(0,0)
p2=Ponto(2,4)
```



42



Método `__init__` com valores default

- Os parâmetros de funções ou métodos podem ter valores default.
- Se na chamada não for especificado valores para estes parâmetros, os valores default serão usados.

```
class NomeDaClasse:
    """Texto com a documentação da classe"""
    def __init__(self, params,...,param_n=default_n):
        i_1
        i_n
```

Os parâmetros default devem ser declarados por último.

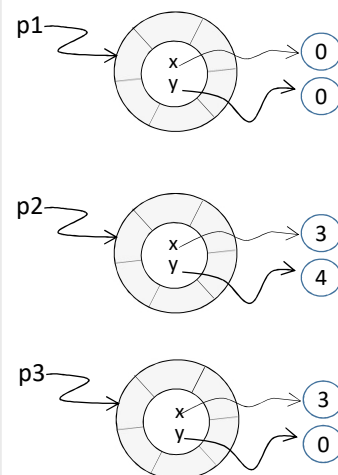
43



Instanciando pontos com valores default

```
class Ponto:
    """ classe Ponto:
        representa e manipula coordenadas x,y
    """
    def __init__(self,x=0,y=0):
        """ cria novo ponto (x,y) """
        self.x = x
        self.y = y
        return

p1=Ponto()
p2=Ponto(3,4)
p3=Ponto(3)
```



44

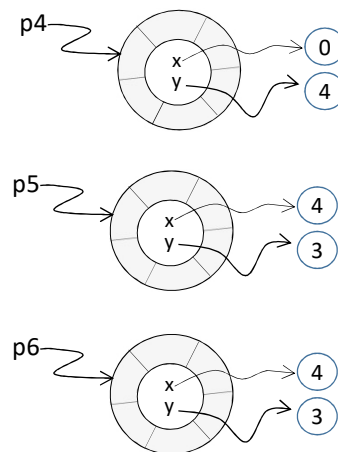


Passando argumentos com nomes

Se na chamada da função ou método, o valor do argumento for identificado com o nome do parâmetro, não é necessário empregar a ordem de definição

```
class Ponto:
    """
    classe Ponto: representa e manipula coordenadas x,y
    """
    def __init__(self, x=0, y=0):
        """ cria novo ponto (x,y) """
        self.x = x
        self.y = y
        return

p4=Ponto(y=4)
p5=Ponto(y=3, x=4)
p6=Ponto(4, y=3)
```



45



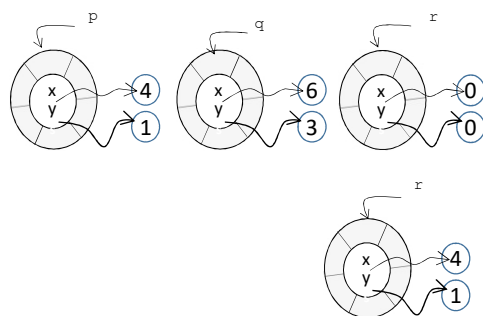
Acessando os atributos

Os atributos de um objeto são acessados usando o operador .

Objeto . atributo

retorna o valor associado ao atributo do objeto

```
p=Ponto(y=1,x=4)
q=Ponto(y=3,x=6)
r=Ponto()
if p.x < q.x:
    r.x = p.x
else:
    r.x = q.x
if p.y < q.y:
    r.y = p.y
else:
    r.y = q.y
```



46



Mãos na massa

Construa uma função que receba um objeto da classe Ponto e retorne o número do quadrante a que pertence:

1º Quadrante = $\{(x, y) \mid x > 0 \text{ e } y > 0\}$

2º Quadrante = $\{(x, y) \mid x < 0 \text{ e } y > 0\}$

3º Quadrante = $\{(x, y) \mid x < 0 \text{ e } y < 0\}$

4º Quadrante = $\{(x, y) \mid x > 0 \text{ e } y < 0\}$

```
class Ponto:
    """
    classe Ponto: representa e manipula coordenadas x,y
    """
    def __init__(self, x=0, y=0):
        """ cria novo ponto (x,y) """
        self.x = x
        self.y = y
        return
```

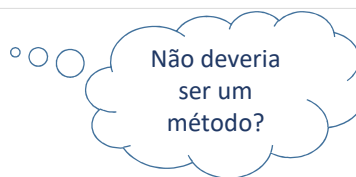
47



Uma Solução

```
def quadrante(p):
    if p.x >= 0:
        if p.y > 0:
            return 1
        elif p.y < 0:
            return 4
        else:
            return 'origem'
    elif p.y > 0:
        return 2
    else:
        return 3
```

```
p5=Ponto(y=3,x=4)
p6=Ponto(y=-3,x=8)
print('Ponto 5 está no quadrante: ',quadrante(p5))
print('Ponto 6 está no quadrante: ',quadrante(p6))
```



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (tags/3.5.2:Mar 28 2016, 22:01:18) [AMD64] on win32
Type "help()" or "license()" for more information.

Ponto 5 está no quadrante: 1
Ponto 6 está no quadrante: 4
```

48



Criando métodos

Métodos são funções definidas dentro de uma classe.

```
class NomeDaClasse:
    """ Texto com a documentação da classe """
    def __init__(self, param,param,...param):
        i_1
        i_n
    def método (self, params,...,param_n=default_n):
        i_1
        i_n
```

self: referência à instância que chamou o método

params: Opcionais. Especificam os parâmetros do método.

Todo o método possui o *self* como primeiro parâmetro:

referência à instância que chama o método - por convenção, self

49



Classe Ponto com método Quadrante

```
class Ponto:
    ''' classe Ponto: representa e manipula coordenadas x,y '''
    def __init__(self,x=0,y=0):
        ''' cria um novo ponto (x,y) '''
        self.x = x
        self.y = y
        return

    def quadrante(self):
        ''' retorna o n° do quadrante de um ponto ou 'origem' '''
        if self.x>=0:
            if self.y>0:
                return 1
            elif self.y<0:
                return 4
            else:
                return 'origem'
        elif self.y>0:
            return 2
        else:
            return 3
```

50

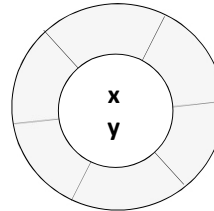


Mãos na Massa

Desenvolvendo a classe ponto

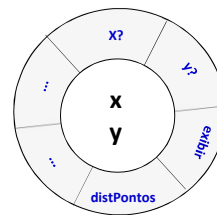
Atributos:

coordenada x
coordenada y



Métodos:

exibir o ponto no formato usual: (x,y)
calcular a distância entre dois pontos
calcular a distância do ponto à origem
exibir o quadrante do ponto
desloca n unidades
....




51



Classe Ponto_{1/2}

```
class Ponto:
    """ classe Ponto: representa e manipula coordenadas x,y """
    def __init__(self,x=0,y=0):
        """ cria um novo ponto (x,y) """
        self.x = x
        self.y = y
        return
    def exibir(self):
        """ exibe o ponto no formato(x,y) """
        print ("(%.2f,%.2f)"%(self.x, self.y))
    def distanciaDaOrigem(self):
        """ calcula distância do ponto à origem """
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5
    def distanciaEntrePontos(self,outro):
        """ calcula distância do ponto à origem """
        return (((self.x-outro.x) ** 2)+((self.y-outro.y)** 2))**0.5
```

52




Classe Ponto _{2/2}

```
def quadrante(self):
    """ retorna o n° do quadrante de um ponto ou 'origem' """
    if self.x>=0:
        if self.y>0:
            return 1
        elif self.y<0:
            return 4
        else:
            return 'origem'
    elif self.y>0:
        return 2
    else:
        return 3

def desloca(self, n):
    """ desloca as coordenadas do ponto n unidades """
    self.x +=n
    self.y +=n
```

53



Ativando os métodos

Os métodos de um objeto são ativados usando o operador `.`

Objeto . método(argumentos)

Executa o método para a instância (objeto) que o invocou

ATENÇÃO:

- Na definição do método, o primeiro parâmetro é sempre o próprio objeto (self)
- Na ativação do método o objeto (self) não é argumento

54



Self não é argumento na ativação

Embora o *self* deva ser explicitamente especificado quando o método é definido, ele não é incluído em sua chamada. O Python o passa automaticamente.

Definição do método na classe:

class Ponto:

```
...
def desloca(self, n):
    self.x += n
    self.y += n
```

Chamada do método:

```
p=Ponto()
>>> p.desloca(9)
```

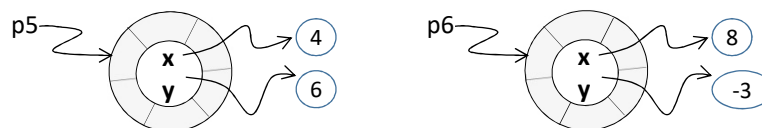
na chamada do método, a instância que o invocou é passada automaticamente na posição do self

55



Solução com a nova classe Ponto

```
p5=Ponto(y=6,x=4)
p6=Ponto(8,-3)
print('Ponto 5 está no quadrante: ',p5.quadrante())
print('Ponto 6 está no quadrante: ',p6.quadrante())
```



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

Ponto 5 está no quadrante: 1
Ponto 6 está no quadrante: 4
```

56

Mãos na Massa

- Escolha as coordenadas de três pontos. Utilizando a classe Ponto, verifique se estes pontos formam um triângulo. Em caso afirmativo mostre:

a) Baricentro G $G \left(\frac{x_A + x_B + x_C}{3}, \frac{y_A + y_B + y_C}{3} \right)$

b) Os pontos P, M, N

c) Perímetro

d) Área:

Área dado os lados:

$$A^2 = S(S - a)(S - b)(S - c)$$

S: semiperímetro

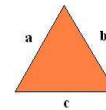
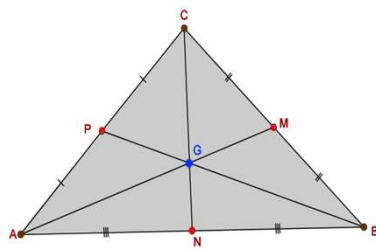
Condição para três pontos formarem um triângulo

A medida de qualquer um dos lados é menor que a soma das medidas dos outros dois e maior que o valor absoluto da diferença entre essas medidas.

$$|b - c| < a < b + c$$

$$|a - c| < b < a + c$$

$$|a - b| < c < a + b$$



57

Uma Solução_{1/3}

```
import math
import ponto
def daCondicao(d1,d2,d3):
    if (d1 > (d2+d3)) or (math.fabs(d2-d3) > d1):
        return False
    return True

def ehTriangulo(pA,pB,pC):
    dAB=pA.distanciaEntrePontos(pB)
    dAC=pA.distanciaEntrePontos(pC)
    dBC=pB.distanciaEntrePontos(pC)
    return( daCondicao(dAB,dAC,dBC) and
            daCondicao(dAC,dAB,dBC) and
            daCondicao(dBC,dAB,dAC) )
```

58



Uma Solução_{2/3}

```
def calcValores (pA, pB, pC) :
    pG=ponto.Ponto ( (pA.x+pB.x+pC.x) /3, (pA.y+pB.y+pC.y) /3)
    pP=ponto.Ponto ( (pA.x+pC.x) /2, (pA.y+pC.y) /2)
    pM=ponto.Ponto ( (pB.x+pC.x) /2, (pB.y+pC.y) /2)
    pN=ponto.Ponto ( (pA.x+pB.x) /2, (pA.y+pB.y) /2)
    dAB=pA.distanciaEntrePontos (pB)
    dAC=pA.distanciaEntrePontos (pC)
    dBC=pB.distanciaEntrePontos (pC)
    perim=dAB+dAC+dBC
    S=perim/2
    area=math.sqrt ( (S*(S-dAB) * (S-dAC) * (S-dBC)) )
    print ("Baricentro:", end=' '); pG.exibir()
    print ("P:", end=' '); pP.exibir()
    print ("M:", end=' '); pM.exibir()
    print ("N:", end=' '); pN.exibir()
    print ("Perim:%.2f Area:%.2f"% (perim, area))
    return
```

59



Uma Solução_{3/3}

```
#Exemplo 1
pA=Ponto (1,2)
pB=Ponto (7,4)
pC=Ponto (5,0)
if ehTriangulo (pA, pB, pC) :
    print ("OK")
    calcValores (pA, pB, pC)
else:
    print ("NOK")

#Exemplo 2
pA=Ponto (-1,1)
pB=Ponto (4,0)
pC=Ponto (-3,3)
if ehTriangulo (pA, pB, pC) :
    print ("OK")
    calcValores (pA, pB, pC)
else:
    print ("NOK")
```

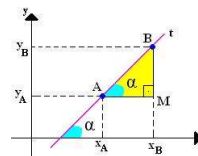
Não seria melhor criar uma
classe triângulo?

60

Exercícios

1) Crie métodos na classe Ponto para:

- 1) determinar o ponto mediano entre dois pontos
- 2) calcular o coeficiente angular da reta que passa pelo ponto e a origem
- 3) calcular os coeficientes angular da reta que passa por dois pontos
- 4) alterar o valor do x ou o valor do y
- 5) retornar o valor do x ou do y
- 6) clonar o ponto



Coeficiente Angular m:

$$m = \operatorname{tg} \alpha = \frac{y_B - y_A}{x_B - x_A}$$

61

Exercício Usando a Classe Ponto

1) Determinar se 3 pontos são colineares :

- o o determinante da matriz quadrada calculado pela regra de Sarrus é igual a 0.

Exemplo: Os pontos A (2, 5), B (3, 7) e C (5, 11) são colineares:

Diagonal principal

$$2 * 7 * 1 = 14$$

$$5 * 1 * 5 = 25$$

$$1 * 3 * 11 = 33$$

Diagonal secundária

$$1 * 7 * 5 = 35$$

$$2 * 1 * 11 = 22$$

$$5 * 3 * 1 = 15$$

$$\begin{vmatrix} 2 & 5 & 1 \\ 3 & 7 & 1 \\ 5 & 11 & 1 \end{vmatrix} = 0$$

$$\begin{vmatrix} 2 & 5 & 1 & 2 & 5 \\ 3 & 7 & 1 & 3 & 7 \\ 5 & 11 & 1 & 5 & 11 \end{vmatrix} = 0$$

Somatório diagonal principal – Somatório diagonal secundária

$$(14 + 25 + 33) - (35 + 22 + 15)$$

$$72 - 72 = 0$$


62



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Métodos Especiais ou "Métodos Mágicos"

63



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Métodos especiais

Os métodos mágicos (**magic methods**), também chamados de métodos **dunderscore** (**double-underscore**) ou de **métodos especiais**, permitem que objetos de classes definidas pelo programador possuam uma interface de acesso semelhante aos objetos nativos da linguagem.

64

Métodos especiais para exibição

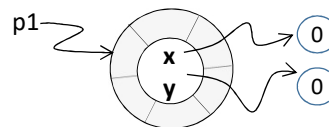
65

Exibindo o objeto com print()

```
class Ponto:
    """ classe Ponto:
    representa e manipula coordenadas x,y """
    def __init__(self, x=0, y=0):
        """ cria um ponto na origem """
        self.x = 0
        self.y = 0
        return

>>> p1=Ponto()
>>> print(p1)
```

<__main__.Ponto object at 0x1006e5b50 >



O comando *print* espera uma string como argumento.

Como usá-lo para exibir um objeto?

método `__str__`

66



O método especial `__str__`

- ✓ O método `__str__`, quando definido, tem como objetivo criar uma representação textual do objeto, definida pelo programador, e retornar a string criada.

```
class NomeDaClasse:
    """Texto com a documentação da classe"""
    ....
    def __str__(self):
        i_1
        i_n
```

O comando `print` e a função `str` invocam este método quando o argumento passado é um objeto (e o método foi definido). Exibem na tela a string retornada.

67




Classe Ponto com método `__str__`

```
class Ponto:
    """ classe Ponto: representa e manipula coordenadas x,y """
    def __init__(self,n,x,y):
        """ cria novo ponto(x,y) identificado por n """
        self.n = n
        self.x = x
        self.y = y
        return
    def __str__(self):
        """ monta string → (x,y) """
        s= " (%f,%f)" %(self.x,self.y)
        return s

    def quadrante(self):
        """ retorna o n° do quadrante de um ponto ou 'origem' """
        ...
```

68

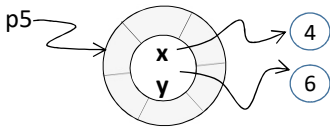
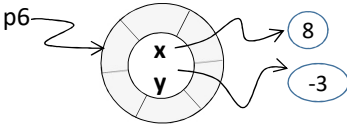


Exibindo um obj Ponto com print

```

p5=Ponto(y=6,x=4)
p6=Ponto(8,-3)
print('P5: ',p5)
print('P6: ',p6)

```

Python 3.5.2 Shell

File Edit Shell Debug Options Window Help


Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32

Type "copyright", "credits" or "license()" for more information.

P5: (4.0,6.0)

P6: (8.0,-3.0)

69



.format() Formatando string

Forma Básica:


' texto{ }texto{ }... '.format(val1,val2,...)

- Constrói-se um esqueleto da string, marcando com {} onde o valor será inserido.
- As chaves e seus conteúdos (chamados de campos de formatação) são substituídos pelos objetos passados para o método [str.format\(\)](#), respeitando a ordem dos argumentos.

Exemplo:

"Hoje, {}/{}/, é Natal".format(25,12,2017) 'Hoje, 25/12/2017, é Natal'

70



.format()

Formatando string

Forma com especificação de formatação:

`' texto{ :formato }texto{ :formato }... '.format(val1,val2,...)`

onde formato pode especificar, entre outros, o alinhamento (e respectivo caractere de preenchimento), a largura (com nº de casas decimais para float) e o tipo;


Algumas opções de alinhamento (c é ' ' por padrão)

Opções	Significado
'c <'	Alinha pela esquerda
'c >'	Alinha pela direita (padrão p/nºs)
'c ^'	Centraliza

Algumas opções de tipo

Opções	Tipo
s	string
d	int
f	float

71



Exemplos

.format com especificação de formatação

```

"Hoje, {}/{} / {}, é Natal".format(25,12,2017)
'Hoje, 25/12/2017, é Natal'
"Hoje, {:4d}/{:4d}/{:8d}, é Natal".format(25,12,2017)
'Hoje,   25/   12/   2017, é Natal'
"Hoje, {:<4d}/{:<4d}/{:<8d}, é Natal".format(25,12,2017)
'Hoje, 25  /12  /2017   , é Natal'
"Hoje, {:>4d}/{:>4d}/{:>8d}, é Natal".format(25,12,2017)
'Hoje,   25/   12/   2017, é Natal'
"Hoje, {:^4d}/{:^4d}/{:^8d}, é Natal".format(25,12,2017)
'Hoje,  25 / 12 / 2017  , é Natal'
"Hoje, {:*^4d}/{:*^4d}/{:*^8d}, é Natal".format(25,12,2017)
'Hoje, *25*/*12*/**2017**, é Natal'
"Hoje, {:*>4d}/{:*>4d}/{:*>8d}, é Natal".format(25,12,2017)
'Hoje, **25/**12/****2017, é Natal'
"Hoje, {:*<4d}/{:*<4d}/{:*<8d}, é Natal".format(25,12,2017)
'Hoje, 25**/12**/2017****, é Natal'

```

72



.format() Identificação posicional dos argumentos

Forma com especificação de formatação:

```
'texto{nº:formato }texto{nº:formato }...'.format(val1,val2,...)
```

O nº identifica o argumento pela sua posição na chamada do método

Exemplos:

```
"Hoje, {0:4d}/{1:4d}/{2:4d}, é Natal".format(25,12,2017)
'Hoje, 25/12/2017, é Natal'
```

```
"Hoje, {1:4d}/{0:4d}/{2:4d}, é Natal".format(25,12,2017)
'Hoje, 12/25/2017, é Natal'
```

73



.format() Identificação dos argumentos por nome

Forma com especificação de formatação:

```
'texto{nome1:formato }texto{nome2:formato }...'.format(nome1=val1,nome2=val2,...)
```

Se argumentos nomeados são passados para o método [str.format\(\)](#), seus valores podem ser identificados pelo nome do argumento


Exemplos:

```
"Hoje, {dia:4d}/{mes:4d}/{ano:4d}, é Natal".format(dia=25,mes=12,ano=2017)
'Hoje, 25/12/2017, é Natal'
```

```
"Hoje, {mes:4d}/{dia:4d}/{ano:4d}, é Natal".format(dia=25,mes=12,ano=2017)
'Hoje, 12/25/2017, é Natal'
```

```
"Hoje, {mes:4d}/{dia:4d}/{0:4d}, é Natal".format(2017,dia=25,mes=12)
'Hoje, 12/25/2017, é Natal'
```

74



.format() Classe Ponto:método __str__


```

class Ponto:
    """ classe Ponto: representa e manipula coordenadas x,y """
    def __init__(self,n,x,y):
        """ cria novo ponto(x,y) identificado por n """
        self.n = n
        self.x = x
        self.y = y
        return
    def __str__(self):
        """ monta string → (x,y) """
        s= "({:4.2f},{:4.2f})".format(self.x,self.y)
        return s

    def quadrante(self):
        """ retorna o n° do quadrante de um ponto ou 'origem' """
        ...

```

75



O método __repr__

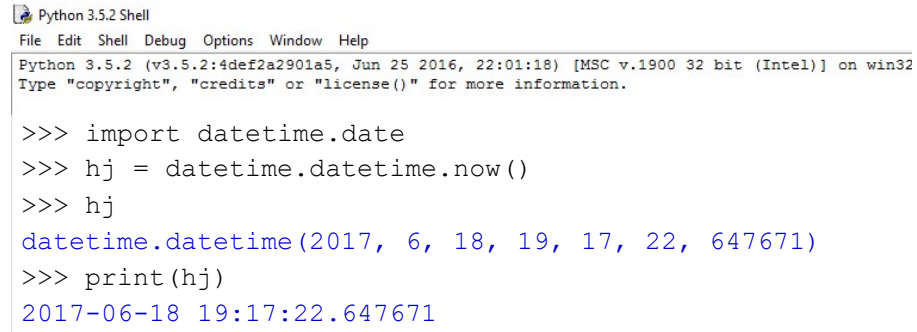
O `__str__` serve para exibir o objeto para o usuário final.

- É usado pelo comando *print* e pela função *str*

O `__repr__`, constrói uma string da representação interna do objeto. Serve para exibir o objeto para o programador.

- É usado pelo *console* do Python e pela função *repr*.

Exemplo:




```

Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

>>> import datetime.date
>>> hj = datetime.datetime.now()
>>> hj
datetime.datetime(2017, 6, 18, 19, 17, 22, 647671)
>>> print(hj)
2017-06-18 19:17:22.647671

```

76



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

.format()


Classe Ponto:método __str__ e __repr__

```

class Ponto:
    """ classe Ponto: representa e manipula coordenadas x,y """
    def __init__(self,n,x,y):
        """ cria novo ponto (x,y) identificado por n """
    def __str__(self):
        """ monta string → (x,y) """
        s = "({:4.2f},{:4.2f})".format(self.x,self.y)
        return s
    def __repr__(self):
        """ representação interna → [x,y] """
        s = "[{:4.2f},{:4.2f}]".format(self.x,self.y)
        return s

```

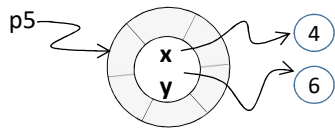
77




DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

__str__ x __repr__

p5=Ponto (y=6,x=4)



 Python 3.5.2 Shell

File Edit Shell Debug Options Window Help


Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

```

>>>p5                                #invoca __repr__
[4.0,6.0]
>>>print (p5)                        #invoca __str__
(4.0,6.0)

```

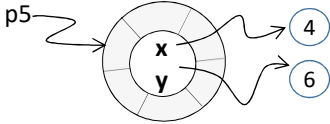
78




__str__ x __repr__ com format

As marcações '!' e '!' podem ser usadas para forçar a conversão de valores aplicando respectivamente aos métodos [str\(\)](#) e [repr\(\)](#)

```
p5=Ponto (y=6, x=4)
```




 Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

```
>>> "{!r}".format(p5)           #invoca __repr__
[4.0,6.0]
>>> "{!s}".format(p5)           #invoca __str__
(4.0,6.0)
```

79



Métodos especiais para operações aritméticas

80



Classes que funcionam como números

Usando os métodos especiais apropriados, pode-se definir a soma, subtração e outras operações matemáticas para a classe.

Operação desejada	Você escreve...	e o Python chama...
Soma	$x + y$	<code>x.<u>__add__</u>(y)</code>
Subtração	$x - y$	<code>x.<u>__sub__</u>(y)</code>
Multiplicação	$x * y$	<code>x.<u>__mul__</u>(y)</code>
Divisão	x / y	<code>x.<u>__truediv__</u>(y)</code>
Floor division	$x // y$	<code>x.<u>__floordiv__</u>(y)</code>
Resto da Divisão	$x \% y$	<code>x.<u>__mod__</u>(y)</code>
Potência	$x ** y$	<code>x.<u>__pow__</u>(y)</code>

81



Exercício

Defina a classe **números binários inteiros** com todas as operações aritméticas (+, -, *, /, %, **, //)

```
>>> bin(12345) # função built-in que converte de int para binário
'0b11000000111001'
>>> int(0b11000000111001) #Converte de binário para int
12345
```

82



Uma Solução

```
class Binario():
    def __init__(self, valor_dec):
        self.valor_dec = valor_dec
        self.valor_bin = bin(self.valor_dec)
    def __str__(self):
        return "%s" % (self.valor_bin[2:]) # para tirar o 0b
    def __add__(self, outrobin):
        return bin(self.valor_dec+outrobin.valor_dec)
    def __sub__(self, outrobin):
        return bin(self.valor_dec-outrobin.valor_dec)
    def __mul__(self, outrobin):
        return bin(self.valor_dec*outrobin.valor_dec)
    def __floordiv__(self, outrobin):
        return bin(self.valor_dec//outrobin.valor_dec)
    def __truediv__(self, outrobin):
        return bin(self.valor_dec/outrobin.valor_dec)
    def __mod__(self, outrobin):
        return bin(self.valor_dec%outrobin.valor_dec)
    def __pow__(self, outrobin):
        return bin(self.valor_dec**outrobin.valor_dec)
```

83



Métodos especiais para atribuições e comparações

84

Atribuição Composta

Operação Desejada	Você escreve...	e o Python chama...
Adição Composta	<code>x += y</code>	<code>x. iadd (y)</code>
Subtração Composta	<code>x -= y</code>	<code>x. isub (y)</code>
Multiplicação Composta	<code>x *= y</code>	<code>x. imul (y)</code>
Divisão Composta	<code>x /= y</code>	<code>x. itruediv (y)</code>
Floor division Composta	<code>x //= y</code>	<code>x. ifloordiv (y)</code>
Resto da Divisão Composto	<code>x %= y</code>	<code>x. imod (y)</code>
Potência Composta	<code>x **= y</code>	<code>x. ipow (y)</code>

85

Comparação

Se na definição da classe, faz sentido comparar objetos, os seguintes métodos especiais podem ser desenvolvidos para implementar comparações.

Operação Desejada	Você escreve...	E o Python chama...
Igualdade	<code>x == y</code>	<code>x. eq (y)</code>
Desigualdade	<code>x != y</code>	<code>x. ne (y)</code>
Menor que	<code>x < y</code>	<code>x. lt (y)</code>
Menor ou Igual a	<code>x <= y</code>	<code>x. le (y)</code>
Maior que	<code>x > y</code>	<code>x. gt (y)</code>
Maior ou igual a	<code>x >= y</code>	<code>x. ge (y)</code>

86



Exercício

Complete a classe binário com os operadores de comparação

Atenção

object.__cmp__(self, other) :

Deve retornar:

- um inteiro negativo se $\text{self} < \text{other}$,
- zero se $\text{self} == \text{other}$, ou
- um número positivo se $\text{self} > \text{other}$.

87



Exercícios

Complete a classe Ponto com as seguintes operações:

- **+**: $\text{ptoA} + \text{ptoB}$ - novo ponto com coordenada x resultante da soma das coordenadas x de ptoA e de ptoB e coordenada y resultante da soma das coordenadas y de ptoA e de ptoB
- **-**: $\text{ptoA} - \text{ptoB}$ - novo ponto com coordenada x resultante da diferença das coordenadas x de ptoA e de ptoB e coordenada y resultante da diferença das coordenadas y de ptoA e de ptoB
- **==**: $\text{ptoA} == \text{ptoB}$ – retorna verdadeiro se as coordenadas x e y são iguais
- **!=**: $\text{ptoA} != \text{ptoB}$ – retorna verdadeiro se pelo menos uma das coordenadas é diferente
- **<**: $\text{ptoA} < \text{ptoB}$ - se a distância do ptoA à origem é menor que a do ptoB
- **>**: $\text{ptoA} > \text{ptoB}$ - se a distância do ptoA à origem é maior que a do ptoB


88



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Relacionamento entre classes

89



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Relacionamentos entre Classes

- Para que objetos se comuniquem eles precisam se relacionar
- Tipos de Relacionamentos:
 - ✓ Associação : “possui um”
 - ✓ Agregação: “é parte de”, “tem um”
 - Composição – “pertence exclusivamente a”
 - ✓ Dependência : “precisa de”
 - ✓ Generalização / Herança: “é um tipo de” .

<http://www.codeproject.com/Articles/330447/Understanding-Association-Aggregation-and-Composit>

90



Alguns Tipos de Relacionamentos entre Classes

- ✓ Associação : ligação entre duas classes que permitem a comunicação entre os objetos. São completamente independentes entre si mas eventualmente estão relacionados.

Exemplo: relação entre professores e alunos. Um aluno pode ter vários professores e um professor pode ter vários alunos. Um não depende do outro para existir

<http://www.codeproject.com/Articles/330447/Understanding-Association-Aggregation-and-Composit>

91




Alguns Tipos de Relacionamentos entre Classes

- ✓ Associação :
 - ✓ Agregação: “é parte de” é uma associação “tem um”. Existe relação de pertinência, mas os objetos são independentes. A classe interna representa um atributo da classe externa, mas faz sentido mesmo quando a classe externa deixa de existir.

Exemplo: a relação entre os professores e os departamentos. Departamentos podem ter vários professores. E o professor só pode estar vinculado a um departamento. Mas eles são independentes.

<http://www.codeproject.com/Articles/330447/Understanding-Association-Aggregation-and-Composit>

92



Alguns Tipos de Relacionamentos entre Classes

✓ Associação :


✓ Agregação:

- Composição – “...pertence exclusivamente a ...” é uma variação de agregação onde há dependência entre os objetos, ou seja, as partes não existem sem o todo. Se o objeto principal for destruído, os objetos que o compõe não podem existir mais.

Exemplo: a relação entre uma universidade e os departamentos. Além da universidade possuir vários departamentos, eles só podem existir se a universidade existir.

<http://www.codeproject.com/Articles/330447/Understanding-Association-Aggregation-and-Composi>

93



Alguns Tipos de Relacionamentos entre Classes

✓ Associação

✓ Dependência : “precisa de” um objeto depende da especificação do outro. As partes do todo geralmente funcionam juntas. Não é possível existir a classe A sem que a classe B já exista.

Exemplo: Uma universidade física depende de uma rua e de um terreno. Qualquer alteração na estrutura do terreno ou da rua pode gerar efeito colateral na universidade.

<http://www.codeproject.com/Articles/330447/Understanding-Association-Aggregation-and-Composi>

94



Alguns Tipos de Relacionamentos entre Classes

- ✓ Generalização / Herança: “é um tipo de”.

Há relação de herança nas propriedades:

A sub-classe herda as propriedades (atributos, métodos e relações) da super-classe, podendo alterar as herdadas ou acrescentar outras.

<http://www.codeproject.com/Articles/330447/Understanding-Association-Aggregation-and-Composit>

95



Exemplificando os relacionamentos

- Um laptop é formado por elementos que se relacionam entre si:
 - ✓ CPU, memória, placa de vídeo, placa mãe, Teclado, Monitor, Cooler, fonte,...
- Esses elementos juntos e corretamente conectados tornam possível a existência do laptop.
- Exemplos de relacionamento entre os componentes de um laptop:
 - Dependência: A CPU depende dos registradores e barras de endereço e dados para executar as instruções,
 - Associação: uma pessoa possui um laptop
 - Agregação: Monitor, teclado, mouse são partes do sistema de comunicação laptop x humanos
 - Generalização: memória principal (RAM, ROM, cache,..) e memória secundária (SSD, HDs, Cds, Pen-Drives, ...) são tipos específicos de memória

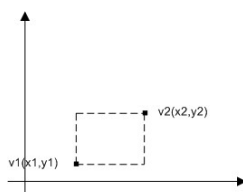
96

Classes Compostas


97

Exercício

Exemplo: Faça um programa que mostre a área de um retângulo a partir dos pontos do vértice inferior esquerdo e do vértice superior direito, fornecidos pelo usuário.

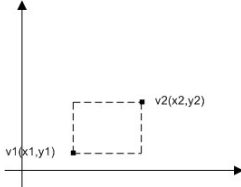


98



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO


Desenvolvendo a solução



Elementos conceituais manipulados pelo programa:

- Ponto
- Retângulo

99



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Como representar o retângulo?

Neste problema, um retângulo é representado por dois pontos:

- ✓ vértice inferior esquerdo
- ✓ vértice superior direito


Como representá-los?

novo tipo composto cujos atributos são objetos do tipo Ponto

Quais operações devem ser definidas?

- a) Consultar/Alterar coordenadas
- b) Exibir vértices
- c) Calcular/Exibir base
- d) Calcular/Exibir altura
- e) Calcular/Exibir perímetro
- f) Calcular/Exibir área ...

100



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Classes Compostas

O que fazer se o pto b for inferior ao a?

```

class Retangulo:
    def __init__(self, a=Ponto(), b=Ponto(1,1)):
        self.vert1 = a
        self.vert3 = b
        self.vert2 = Ponto(self.vert3.x, self.vert1.y)
        self.vert4 = Ponto(self.vert1.x, self.vert3.y)

    def base(self):
        return self.vert1.distanciaEntrePontos(self.vert2)

    def altura(self):
        return self.vert1.distanciaEntrePontos(self.vert4)


    def area(self):
        return self.altura()*self.base()

    def __str__(self):
        return ("Vértice Inf Esq: {}\nVértice Sup Dir: {}\nLargura: {}\nAltura: {}".format(
            self.vert1, self.vert2, self.base(), self.altura()))
  
```

Acessa atributo y do objeto Ponto, atributo do obj. Retângulo

Como melhorar a saída? Como "esconder" que a altura e a base são calculadas por métodos e não atributos?

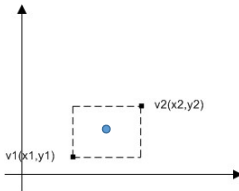
101



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Exercício

Faça um programa que mostre as coordenadas do ponto do centro de um retângulo a partir dos pontos do vértice inferior esquerdo e do vértice superior direito, fornecidos pelo usuário.



102



Exercício

Faça um programa para emitir o carnê de pagamento de uma compra parcelada sem entrada.

A data da compra, o valor a ser parcelado e o número (n) de parcelas será perguntado ao usuário.

Em cada boleto deverá constar o número do boleto (1 a n), a data do vencimento e o valor do boleto.

O intervalo entre as parcelas é de 20 dias.

Crie a classe data

Crie a classe boleto

Resolva o problema

103



Exemplo

Faça um programa para emitir o carnê de pagamento de uma compra parcelada sem entrada.

A data da compra, o valor a ser parcelado e o número (n) de parcelas será perguntado ao usuário.

Em cada boleto deverá constar o número do boleto (1 a n), a data do vencimento e o valor do boleto.

O intervalo entre as parcelas é de 20 dias.



104



Desenvolvendo a Solução

Na Programação Orientada a Objetos, o programador é responsável por :



- ✓ identificar os elementos conceituais que o problema lida (objetos),
- ✓ quais dados os representam (atributos),
- ✓ que mensagens podem receber (métodos) e quais ações devem realizar ao receber cada mensagem (comportamento)

Cada definição de objeto corresponde a algum objeto ou conceito do mundo real e as funções que operam com aqueles objetos correspondem à maneira como os objetos do mundo real interagem.

105



Desenvolvendo a Solução: Identificação dos objetos:

Faça um programa para emitir o carnê de pagamento de uma compra parcelada sem entrada.

A data da compra, o valor a ser parcelado e o número (n) de parcelas será perguntado ao usuário.

Em cada boleto deverá constar o número do boleto (1 a n), a data do vencimento e o valor do boleto.

O intervalo entre as parcelas é de 20 dias.


Quais os elementos conceituais deste problema?

Data da Compra

Boleto

Carnê

106



Desenvolvendo a Solução Data da Compra

O que é uma data?

Data: __/__/__


Tipo de dado composto por três atributos:

- dia
- mês
- ano

Quais operações um objeto deste tipo pode realizar?

- Exibir valores dos atributos;
- Validar valores dos atributos ;
- Consultar/Alterar atributos
- Calcular a data antes ou após X dias
- Calcular a diferença de dias entre duas datas
- Descobrir o dia da semana da data; ...

107



Desenvolvendo a Solução Representando a Data da Compra

Uma Lista? Uma string? Uma tupla? Três inteiros?

Um novo tipo definido pelo programador com três valores e operações de manipulação?

Criar uma Classe Data: estrutura de uma data + operações de manipulação
atributos relevantes + comportamento (métodos)

e uma instância desta classe (objeto): para associar à data da compra.

Neste objeto, instância da classe Data, os valores da data da compra são associados aos atributos (estado) e o conjunto de operações definidas (métodos) pode ser ativado

Classe

Data	
dia	<div style="display: flex; align-items: center;"> <div style="color: red; font-weight: bold; margin-right: 5px;">atributos</div> <div style="font-size: 2em;">{</div> </div>
mes	
ano	
MudaDia()	<div style="display: flex; align-items: center;"> <div style="color: red; font-weight: bold; margin-right: 5px;">métodos</div> <div style="font-size: 2em;">{</div> </div>
DiaSemana()	
DiasDecorridos()	
DataInterv()	
...	

Objeto

dia=9
mes=8
ano=2016
MudaDia()
DiaSemana()
DiasDecorridos()
DataInterv()
...

108



Desenvolvendo a Solução

O que é um boleto?

Tipo de dado composto por 3 atributos:

- Número do boleto
- Data do Vencimento
- Valor do Pagamento



Quais operações um objeto deste tipo pode realizar?

- Exibir valores dos atributos;
- Validar valores dos atributos ;
- Alterar data de Vencimento;
- Calcular juros;
- ...

109



Desenvolvendo a Solução Representando os boletos

Uma lista? Uma tupla? Um dicionário?

Um novo tipo definido pelo programador com três valores e operações de manipulação?

Criar uma Classe Boleto: atributos relevantes + métodos

e instâncias desta classe (objetos): para associar os dados dos boletos.

Classe


Boleto
nBoleto
valor
dtVenc
MudaDtVenc()
Exibir()
MudaValor()

Objetos

n=1	valor=95.00	dtVenc=	dia=29	mes=8	ano=2016	Exibr()
MudaDtVenc()						
Exibir()						
MudaValor()						

n=2	valor=95.00	dtVenc=	dia=18	mes=9	ano=2016	Exibr()
MudaDtVenc()						
Exibir()						
MudaValor()						

110



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO


Desenvolvendo a Solução Carnê

O que é um carnê?


- Lista de boletos

Quais operações um objeto deste tipo pode realizar?

- Incluir boleto;
- Excluir boleto;
- Exibir boleto;
- ...



111



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Desenvolvendo a Solução: Representando o Carnê

Uma Lista? Um dicionário?

Um novo tipo definido pelo programador com três valores e operações de manipulação?

O carnê como uma lista:

n=1
valor=95.00
dtVenc= {
 dia=29
 mes=8
 ano=2016
 Exibr()

}

MudaDtVenc()
Exibir()
MudaValor()
...

n=2
valor=95.00
dtVenc= {
 dia=18
 mes=9
 ano=2016
 Exibr()

}

MudaDtVenc()
Exibir()
MudaValor()
...

n=3
valor=95.00
dtVenc= {
 dia=8
 mes=10
 ano=2016
 Exibr()

}

MudaDtVenc()
Exibir()
MudaValor()
...

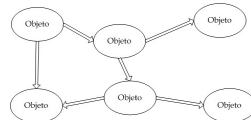
112



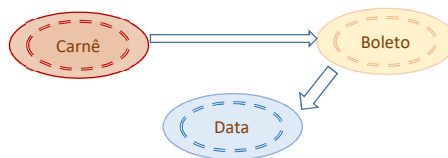
Orientação a Objetos em Programação

Um programa é uma coleção de objetos que interagem, se comunicam para a solução do problema.

A maioria das computações é expressa em termos de operações sobre objetos.



A comunicação entre os objetos é feita através do envio de mensagens uns aos outros com o objetivo de realizar alguma tarefa dentro do programa.



113

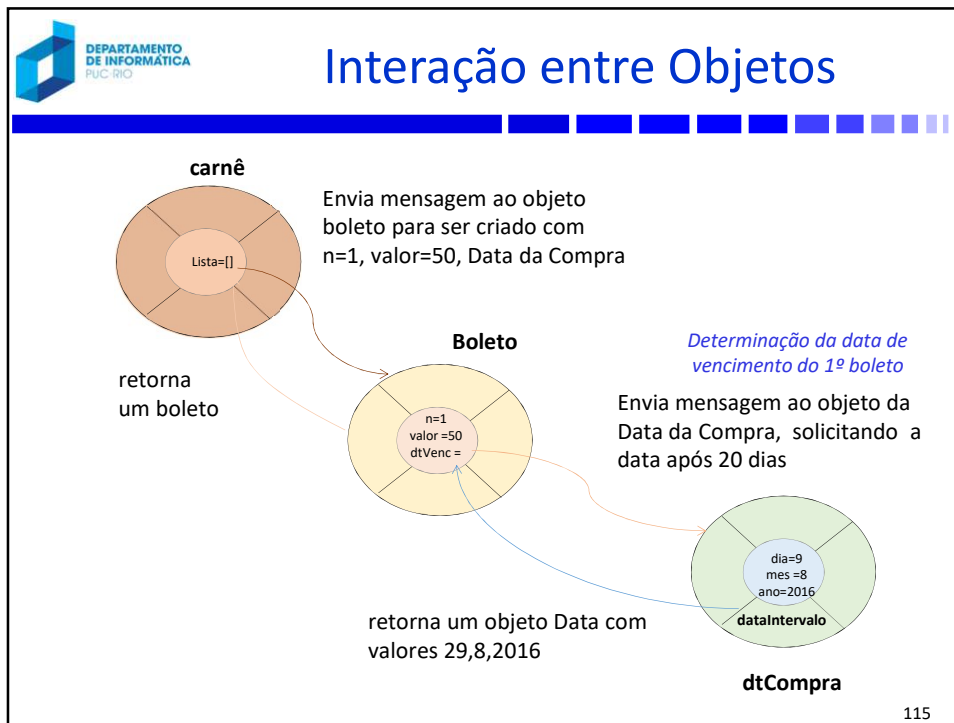


Desenvolvendo a Solução

```

Perguntar dia, mês e ano    →   instanciar data da compra
Perguntar valor total
Perguntar n° de parcelas
parcela = valor total / n° de parcelas
x = 1
carnê = []                  →   instanciar lista vazia compra
Enquanto x < n° de parcelas
    Criar boleto com
        N° boleto = x        →   instanciar boleto
        Data do Vencimento = data da compra + 20 * n° boleto
        Valor do Pagamento = parcela
    Incluir boleto no carnê
Exibir dados dos boletos no carnê
  
```

114



DEPARTAMENTO DE INFORMÁTICA PUC RIO

Exercício

Desenvolva as classes boleto e data

Construa o programa para a emissão do carnê

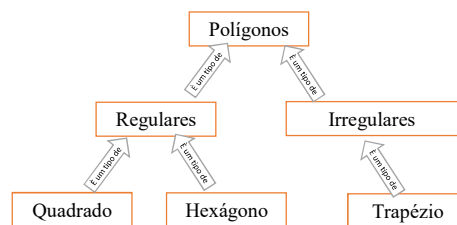
116

Herança

117

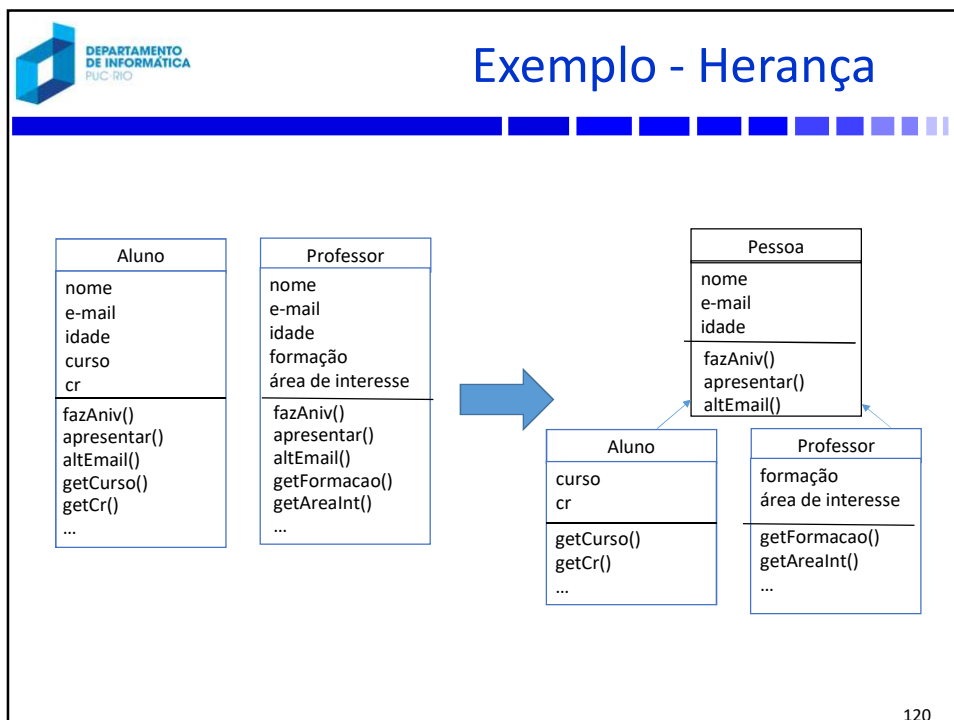
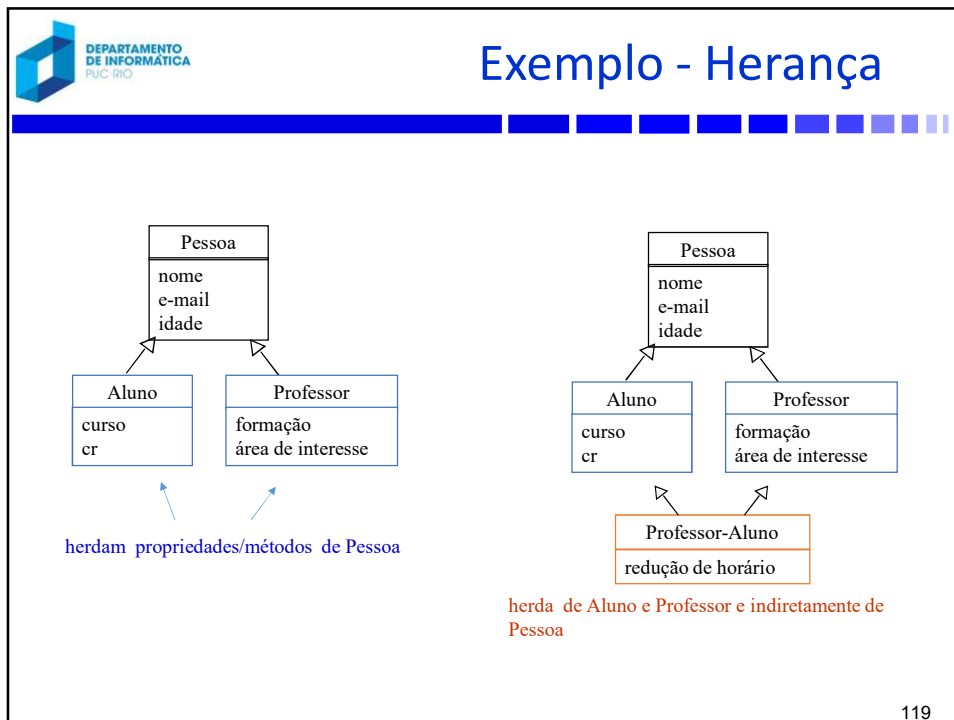
Generalização (Herança)


- Classes podem ser definidas a partir de outras classes (superclasse):
 - ✓ toda a subclasse herda os estados e os comportamentos definidos na superclasse
 - ✓ as subclasses podem modificar ou acrescentar estados e comportamentos herdados



Permite que classes semelhantes compartilhem atributos e/ou métodos

118






Definindo a classe derivada

```
class Nomedaclass(base_1,base_2,...,base_n):
    """Texto com a documentação da classe"""
    #corpo da classe
    atr = valor
    ...
    atr = valor
    def metodo (self, ... param):
        ...
    def metodo (self, ... param):
        ...
```

base_1,...,base_n: classes base

121



O método especial `__init__` na classe derivada

- Pode ser omitido se for exatamente igual ao da classe base.
- A função **`super()`** pode ser utilizada para invocar **qualquer** método da classe base

```
class Nomedaclass(base):
    """Texto com a documentação da classe"""
    def __init__(self, params):
        super().__init__(params)
        i_1
        i_n
```

`base.__init__(params)`

`self`: referência à instância que chamou o método
`params`: Opcionais. Especificam os parâmetros do método

122



Exemplo

```
class Retangulo:
    def __init__(self, lado1=1, lado2=1):
        self.lado1 = lado1
        self.lado2 = lado2
    def area(self):
        return self.lado1 * self.lado2
    def perimetro(self):
        return 2 * (self.lado1 + self.lado2)

class Quadrado(Retangulo):
    def __init__(self, lado=1):
        super().__init__(lado, lado)

print('Perímetro do Retângulo: ', Retangulo(10,20).perimetro())
print('Perímetro do Quadrado: ', Quadrado(10).perimetro())
```

Perímetro do Retângulo: 60
Perímetro do Quadrado: 40

123



Redefinindo métodos da classe base

Qualquer método da classe base pode ser redefinido na classe derivada

Alt: 10 Compr: 20 Lado: 10

```
class Retangulo:
    def __init__(self, lado1=1, lado2=1):
        self.lado1 = lado1
        self.lado2 = lado2
    def exibeMedidas(self):
        print('Alt: {} Compr: {}'.format(self.lado1, self.lado2))
    def area(self):
        return self.lado1 * self.lado2
    def perimetro(self):
        return 2 * (self.lado1 + self.lado2)

class Quadrado(Retangulo):
    def __init__(self, lado=1):
        super().__init__(lado, lado)
    def exibeMedidas(self):
        print('Lado: {}'.format(self.lado1))
        return

Retangulo(10,20).exibeMedidas()
Quadrado(10).exibeMedidas()
```

124



Ativando métodos da classe base

Qualquer método da classe base pode ser ativado

```
super().método(params)
```

OU

```
base.método(params)
```

125



Exemplo


```
class Produto(object):
    def __init__(self, tipo, peso, preco, fabricante):
        self.peso = peso
        self.preco = preco
        self.fabr = fabricante
        self.tipo = tipo
    def __str__(self):
        return 'Produto: {} fabricado por {} \nPreço: R${:.2f} \nPeso: {:.2f} '.format(self.tipo, self.fabr, self.preco, self.peso)

class Computador(Produto):
    def __init__(self, tipo, fabricante, peso, preco, tamanhoTela):
        super().__init__(tipo, peso, preco, fabricante)
        self.tamTela = tamanhoTela
    def __str__(self):
        return super().__str__() + '\nTamanho da Tela: {}'.format(self.tamTela)

lap = Computador('Notebook', 'Dell', 1.4, 3000, 15)
print(lap)
```

Produto: Notebook fabricado por Dell
Preço: 3000.00
Peso: 1.40
Tamanho da Tela: 15

126




Funções Úteis

Hierarquia de uma classe e instâncias:

isinstance(objeto, classe): verifica se o objeto passado é uma instância da classes

issubclass(classe_a, classe_b): verifica se classe_a é uma sub-classe de classe_b

127




Funções Úteis

- **hasattr(objeto, atributo):** verifica se um objeto possui um atributo.

```
>>> hasattr(f1, lado_a)
True
>>> hasattr(f1, lado)
False
```

128




Orientação a Objetos

Introspecção e reflexão: Python permite obter, em tempo de execução, informações a respeito do tipo dos objetos, incluindo informações sobre a hierarquia de classes.

- **dir(objeto):** permite conhecer todos os atributos e métodos de uma classe ou instância.

```
>>> from formas import *
>>> r = Quadrado(13)
>>> dir(r)
['__doc__', '__init__', '__module__', 'calcula_area',
'calcula_perimetro', 'lado_a', 'lado_b']
```

129



Orientação a Objetos

- **__dict__:** apresenta um dicionário com todos os atributos de uma instância.


```
>>> r.__dict__
{'lado_a': 13, 'lado_b': 13}
```

130



Visibilidade e Encapsulamento (Tópicos Complementares)

131



Problema 1

Data
dia
mes
ano
era
DiaSemana()
DiasDecorridos()
DataInterv()
...

Esta classe possui quatro atributos:


- dia: aceita valores entre 1 e 31
- mês: aceita valores entre 1 e 12
- ano: aceita valores ≥ 0
- era: 'aC' ou 'dC'.

```

Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

>>> nascNewton=Data(4,1,1643,'dC')
>>> nascNewton.dia=40      # Inconsistente!!!
  
```

132




Problema 2

Relógio
hora
min
seg
ajustaRelógio ()
decrementaSeg()
incrementaSeg()
...

Esta classe possui três atributos:

- hora: aceita valores entre 0 e 23
- min: aceita valores entre 1 e 59
- seg: aceita valores entre 1 e 59

 Python 3.5.2 Shell

File Edit Shell Debug Options Window Help


Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

```

>>> r1=Relógio(14,0,10)
>>> r1.hora=40 # Inconsistente!!!

```

133




Problema 3

Retângulo
compr
larg
area
....
...

Esta classe possui três atributos:

- compr: valor > 0
- larg: valor > 0
- area: compr*larg

 Python 3.5.2 Shell

File Edit Shell Debug Options Window Help


Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

```

>>> r1=Retângulo(2,3)
>>> r1.area=40 # Inconsistente!!!

```

134




Problema 4

Pessoa
nome sobrenome
nomeCompleto ...

Esta classe possui dois atributos:


- nome:
- sobrenome:
- Método: NomeCompleto

 Python 3.5.2 Shell

```
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

>>> p1=Pessoa('Donald','Patinhas')
>>> r1.Nome
Donald
>>> r1.nomeCompleto()      # expõe a implementação
Donald Patinhas
```

135



Visibilidade

Um atributo ou método de um objeto pode ou não estar visível fora da definição da classe.

O tipo de visibilidade do atributo ou método indica o nível de sua acessibilidade.

Principais tipos de visibilidade:

- ✓ Público: O atributo ou método pode ser utilizado em qualquer lugar (por objetos de qualquer classe);
- ✓ Protegido: O atributo ou método pode ser utilizado no local de sua definição e nas classes filhas (que herdam) da classe onde foi definido (por objetos da classe que o definiu e de suas sub-classes);
- ✓ Privado: O atributo ou método só pode ser utilizado no local de sua definição (por objetos da classe que o definiu)
- ✓ Pacote: Todas as classes do mesmo pacote da classe que definiu o atributo ou método podem acessá-los

136



Visibilidade em Python

Em Python, **todos** os atributos e métodos são **públicos**.

Convenções para limitar o acesso a um atributo ou método:

- **atributos ou métodos iniciados por `_`:**

notação usada por programadores Python para indicar que o acesso ao atributo/método não é aconselhável. Não impede que o atributo/método seja acessado ou modificado de fora do objeto

- **atributos ou métodos iniciados por `__`:**

O acesso direto ao método/atributo retorna um erro de atributo não encontrado. A atribuição de um valor não altera o atributo e não retorna mensagem de erro. Nesta convenção, o interpretador Python modifica o nome do atributo/método do seguinte modo:

`obj._Classe__Atributo/__Método.`

Para invocar o acesso de dentro da classe: `obj.__Atributo/__Método`

de fora da classe: `obj._Classe__Atributo/Método.`

137



Exemplo 1/2


```
class Relogio:
    def __init__(self, h=0, m=0, s=0):
        self.seg = 0
        self._min = 0
        self.__hora = 0
    def __str__(self):
        return "{:02d}:{:02d}:{:02d}h".format(self.__hora, self._min, self.seg)

Relogio = Relogio()
print("Hora Inicial do relógio: ", relogio)

# Alterando os segundos e os minutos
relogio.seg = 85
relogio._min = 85
print("Cuidado minutos alterados diretamente. Hora do relógio: ", relogio)

# Alterando a hora via nome modificado do atributo
relogio._Relogio__hora += 1
print("Hora foi modificada. Hora do relógio: ", relogio)
```

138



Exemplo 2/2

```
#Acessando diretamente a hora: Erro
print(relogio.__hora*3)

#Tentativa de alterar a hora
relogio.__hora = 3  ➡ Atribuição num atributo inexistente, cria um novo atributo no objeto
print("'Não dá erro mas a hora não foi modificada. Hora do relógio: '",
      relogio)
```

Python 3.5.2 Shell


File Edit Shell Debug Options Window Help

Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
 Type "copyright", "credits" or "license()" for more information.

Traceback (most recent call last):
 File ... in <module>
 print(relogio.__hora*3)
 AttributeError: 'Relogio' object has no attribute '__hora'

Não dá erro mas a hora não foi modificada. Hora do relógio: 01:01:01h


139



Encapsulamento

Os objetos devem ser independentes e esconder detalhes de sua implementação ("caixa preta").

O encapsulamento é um princípio da OO, que protege as informações restringindo o acesso a elas: os atributos de um objeto não devem ser acessados diretamente.



Métodos *getters* e *setters* são criados para prover acesso externo a atributos de objetos:

- ✓ getters: servem para se acessar o valor de determinado atributo
- ✓ setters: servem para mudar o valor de determinado atributo, mantendo sua integridade.

Através da interface, é exposto o que o objeto faz, e não como ele faz, nem o que ele tem.

140



Exemplo Data: getters e setters_{1/2}

```
class Data:
    """ classe Data: representa e manipula datas """
    def __init__(self, d=1, m=1, a=0):
        if not self.validoDiaMesAno(d, m, a):
            print("Erro nos valores")
            d=m=a=1
        self.dia = d
        self.mes = m
        self.ano = a
        return

    def __str__(self):
        """ monta string → dd/mm/aaaa """
        return
        "{:02d}/{:02d}/{:04d}".format(self.dia, self.mes, self.ano)
```

141



Exemplo Data: getters e setters_{2/2}

```
def setDia(self, d=1):
    """ se válido, altera o dia para valor recebido """
    if self.validoDiaMesAno(d, self.mes, self.ano):
        self.dia=int(d)
    return

def getDia(self):
    """ retorna o dia como 'dd' """
    return "{:02d}".format(self.dia)

def setMes(self, m=1):
    """ se válido, altera o mes para valor recebido """
    if self.validoDiaMesAno(self.dia, m, self.ano):
        self.mes=int(m)
    return

def getMes(self):
    """ retorna o mês como 'mm' """
    return "{:02d}".format(self.mes)

def setAno(self, a=0):
    """ se válido, altera o ano para valor recebido """
    if self.validoDiaMesAno(self.dia, self.mes, self.ano):
        self.ano=int(a)
    return

def getAno(self):
    """ retorna o ano como 'aaaa' """
    return "{:04d}".format(self.ano)
```

142



Exemplo Relógio: getters e setters 1/2

```
class Relogio:
    def __init__(self):
        self.setSeg(s)
        self.setMin(m)
        self.setHora(h)

    def __str__(self):
        """ monta string → hh:mm:ss """
        return "{:02d}:{:02d}:{:02d}h".format(self.hora, self.min, self.seg)

    def setHora(self, h=0):
        """ Atribui a hora recebida ou 0 se for inválida """
        if h < 0 or h > 23:
            h = 0
        self.hora = h

    def getHora(self, h=0):
        return self.hora
```

143



Exemplo Relógio: getters e setters 2/2


```
def setMin(self, m=0):
    """ Atribui o minuto recebido ou 0 se for inválido """
    if m < 0 or m > 59:
        m = 0
    self.minut = m

def getMin(self):
    return self.minut

def setSeg(self, s=0):
    """ Atribui o segundo recebido ou 0 se for inválido """
    if s < 0 or s > 59:
        s = 0
    self.seg = s

def getSeg(self):
    return self._seg
```

144



Exemplo

Python 3.5.2 Shell

File Edit Shell Debug Options Window Help


Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

```

>>> r = Relogio(10,10,34)
>>> r.getHora()
10
>>> r.setMin(23)
>>> print(r)
10:23:34

```

145



Problemas com Getters e Setters

1. Forma distintas de acessar diferentes atributos ☹️
 ✓ *os que são protegidos e os que não são*

2. Mudar o acesso dos atributos para getters em setters de uma classe pronta ou em desenvolvimento, implica na modificação de uma quantidade enorme de código ☹️

<http://cs.umbc.edu/courses/331/current/code/python/box.py>

146



Atributos x Getters e Setters x Property

A função `property()` permite que o acesso a atributos seja implicitamente realizado por métodos.

É usada no corpo de uma declaração de classe com a forma:

```
atributo = property(fget, fset, fdel, doc)
```

onde

- ✓ *fget*, *fset*, *fdel* são métodos para acessar, alterar e remover o *atributo*
- ✓ *doc* é uma docstring para o atributo

Não fornecer o setter torna o atributo apenas de leitura (read-only attribute)

147



Exemplo Relógio: getters e setters ^{1/2}


```
class Relogio:
    def __init__(self):
        self.setSeg(s)
        self.setMin(m)
        self.setHora(h)

    def __str__(self):
        """ monta string → hh:mm:ss """
        return '{:02d}:{:02d}:{:02d}h'.format(self._hora, self._minuto, self._seg)

    def _setHora(self, h=0):
        """ Atribui a hora recebida ou 0 se for inválida """
        if h < 0 or h > 23:
            h = 0
        self._hora = h

    def _getHora(self):
        return self._hora
```

148



Exemplo Relógio: getters e setters 1/2

```

def  _setMin(self,m=0):
    ''' Atribui o minuto recebido ou 0 se for inválido '''
    if m<0 or m>59:
        m=0
    self._minuto=m

def  _getMin(self):
    return self._minuto


def  _setSeg(self,s=0):
    ''' Atribui o segundo recebido ou 0 se for inválido '''
    if s<0 or s>59:
        s=0
    self._seg=s

def  _getSeg(self):
    return self._seg


hora=property(fget=_getHora,fset=_setHora)
minut=property(fget=_getMin,fset=_setMin)
seg=property(fget=_getSeg,fset=_setSeg)

```

149



Exemplo

 Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

```

>>> r = Relogio(10,10,34)
>>> r.hora
10
>>> r.minut=23
>>> print(r)
10:23:34

```

150



Decorators @

Um decorator é uma forma de adicionar funcionalidades a funções/métodos/classes, sem alterar seus códigos.

São açúcares sintáticos.

Definição [Wiki do Python](#):

Decorators alteram dinamicamente a funcionalidade de uma função, método ou classe, sem uso direto de subclasses ou alteração do código fonte da função "decorada".

Há os decoradores *built-in* e os criados pelo programador

Um *decorator* é identificado pelo caractere @

- ✓ @property \leftrightarrow *fget*
- ✓ @*atributo.setter* $\leftarrow \rightarrow$ *fset*
- ✓ @*atributo.Deleter* $\leftarrow \rightarrow$ *fdel*