UniCEUB – Bacharelado em Ciência da Computação
Python – Programação Orientada a Objetos
Professor: **Barbosa**

- Classes e Objetos - Fundamentos

- Programação orientada a objetos

O Python é uma linguagem de programação orientada a objetos, o que significa que ela possui recursos que dão suporte à Programação Orientada a Objetos (**POO**).

A programação orientada a objetos tem suas origens na **década de 1960**, mas somente em meados da **década de 1980** ela se tornou no principal paradigma de programação utilizado na criação de software.
O **paradigma** foi desenvolvido como forma de tratar o rápido **aumento no tamanho e complexidade** dos sistemas de software e **facilitar a modificação** desses sistemas grandes e complexos ao longo do tempo.

- Programação procedural versus programação orientada a objetos

No **paradigma de programação procedural** o foco é na escrita de **funções** ou procedimentos que operam sobre os dados.

Na **programação orientada a objetos** o foco é na criação de **objetos** que contém tanto os dados quanto as funcionalidades.

A **definição** de cada objeto corresponde a algum objeto ou conceito no mundo real e as funções que operam sobre tal objeto correspondem as formas que os objetos reais interagem.

- Uma mudança de perspectiva

Em Programação **procedural**:
Escrevemos funções e as chamamos usando uma sintaxe como calcule_soma(v1, v2).
Isso sugere que a **função** é um agente **ativo**, que diz algo como:
“Ei, função calcule a coma! Aqui estão os valores para você calcular.“

Em programação **orientada a objetos**, os **objetos** são os **agentes ativos**.
Transferir a responsabilidade da função para um **objeto** torna possível escrever funções mais versáteis e facilita a manutenção e reuso do código.

A **vantagem** mais importante do estilo de orientação à objetos é que ele é mais **adequado** ao nosso processo mental de agrupamento e mais perto da nossa experiência do mundo real.

No **mundo real** nosso método para cozinhar é parte do nosso forno de micro-ondas — nós não temos uma função cozinhar guardada na gaveta do armário que nós passamos para o micro-ondas. Da mesma forma, usamos métodos do telefone celular para enviar um SMS, ou mudar o seu estado para silencioso.

As **funcionalidades** de um objeto do mundo real tendem a ser intrínsecas a esse **objeto**.
A POO nos permite representar essas funcionalidades com precisão ao organizar nossos programas.

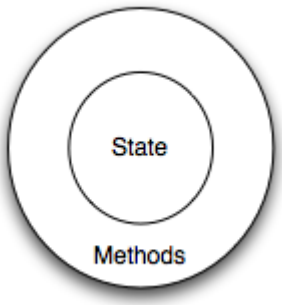
- Objetos

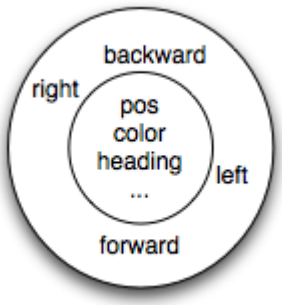
Em Python, todo valor é na verdade um objeto. Seja uma lista, ou mesmo um inteiro, todos são objetos.
Programas manipulam esses objetos realizando computações diretamente com eles ou chamando os seus **métodos** (ou seja, pedindo que esses objetos executem seus métodos).

- Um **objeto** possui:
um **estado** e
uma coleção de **métodos** que ele pode executar.
O estado de um objeto representa as coisas que o objeto sabe sobre si mesmo.

Exemplo:
O objeto **tartaruga**:
Cada tartaruga possui um **estado** que representa a sua posição, sua cor, sua direção etc.
Cada **tartaruga** também tem a **capacidade** (**funcionalidade**) de se mover para a frente, para trás, ou virar para a direita ou esquerda.

Cada tartaruga é diferente pois, embora sejam todas tartarugas, cada uma tem um estado diferente (como posições **diferentes**, ou orientações etc.).





Um objeto simples possui um estado e métodos

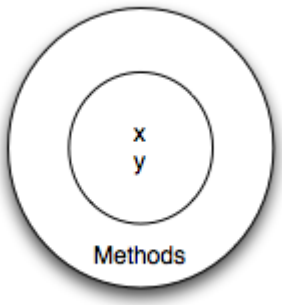
- Classes definidas pelo usuário
Nós já vimos **classes** como str, int, float etc.
Essas foram definidos pelo **Python** e disponibilizadas para nós usarmos.

Precisamos criar **objetos** de dados relacionados ao problema que estamos resolvendo.
Precisamos criar nossas próprias **classes**.

Exemplo:
O conceito de um **ponto matemático**.
Em duas dimensões, um ponto é representado por **dois números** (coordenadas) que são tratados em conjunto como um único objeto.

Os pontos são muitas vezes escritos entre parênteses com uma vírgula separando as coordenadas.
Exemplo:
(0, 0) representa a origem, e
(x, y) representa o ponto x unidades para a direita e y unidades para cima da origem.
Este (x, y) é o estado do ponto.

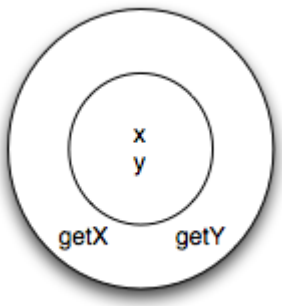
Usando o **diagrama**, desenhamos um objeto ponto:



Um ponto **possui** um x e um y (estado).

Algumas das **operações típicas** que se associa com pontos poderiam ser:
pedir o ponto pela sua coordenada x, get_X,
pedir a sua coordenada y, get_Y.

- Você também pode **calcular**:
A distância de um ponto a partir da origem, ou
A partir de outro ponto, ou
Encontrar um ponto médio de dois pontos, ou
Perguntar se um ponto está dentro de um dado retângulo ou círculo.



- Definindo uma classe em Python
Os pontos tem um **atributo** x e um y, assim a nossa primeira definição da classe fica assim.

- Sintaxe:
class NomeClasse: # Convenção: a primeira letra de cada palavra maiúscula, sem sublinha

    def __init__(self [, parâmetros]): # Método especial construtor

```
        self.atributo_1 = valor_inicial / parametro1          # Atributos (estado)
        self.atributo_2 = valor_inicial / parametro 2
        ...
        self.atributo_n = valor_inicial / parametro n
    def outro_metodo(self [, parâmetros] ):
        pass
```

Exemplo1:

```
class Point:
    """ Point class for representing and manipulating x,y coordinates. """

    def __init__(self):
        """ Create a new point at the origin """
        self.x = 0
        self.y = 0
```

Exemplo2:

```
class Point:
    def __init__(self , x, y):                    # Método especial construtor

        self.x = x
        self.y = y
```

- As definições de classes
Podem aparecer em qualquer lugar em um programa, mas são geralmente perto do **início** (após os comandos import).

As regras de sintaxe para a **definição** de uma classe são as mesmas de outros comandos compostas:
- Um cabeçalho que começa com a palavra-chave **class**,
- Seguido pelo **nome da classe**, e
- Terminando com dois **pontos**.

Se a primeira linha após o cabeçalho de classe é um string, ele se torna o docstring da classe, e será reconhecido por diversas ferramentas (**docstrings** funciona com funções também).

- Método especial ou mágico ou inicializador ou dunders:
Toda classe deve ter um **método** com o nome __init__.

Esse método de **inicialização**, muitas vezes referido como o **construtor**, é chamado automaticamente sempre que uma nova instância de Point é criada.

Ela dá ao programador a oportunidade de configurar os atributos necessários dentro da nova instância, dando-lhes seus estados (valores **iniciais)**.

O parâmetro **self** (pode escolher qualquer outro nome) é definido automaticamente para **referenciar** o objeto recém-criado que precisa ser inicializado.

Então, vamos usar a nossa nova **classe** Point agora.
class Point:
    """ Point class for representing and manipulating x,y coordinates. """

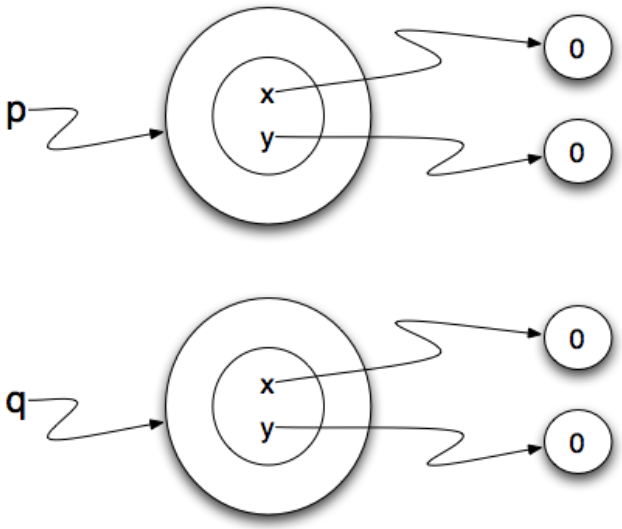
```
    def __init__(self):            # Método construtor
        self.x = 0
        self.y = 0

if __name__ == '__main__':
    p = Point()                    # Instantiate an object of type Point
                                   # Point(), chama o método construtor __init__ da classe.
    q = Point()                    # and make a second point

    print("Nothing seems to have happened with the points")
```

Na **inicialização** dos objetos, criamos dois atributos chamados x e y, ambos com o valor 0.
Quando você executar o programa, nada acontece. Felizmente este não é bem o caso.

Dois Points foram criados, cada um com coordenadas x e y com valor 0.
No entanto, como não pedimos aos pontos para fazerem alguma coisa, não vemos resultado algum.



Glossário

atributo
Um dos itens nomeados de dados que compõem uma instância.

classe
Um tipo de composto definido pelo usuário. Uma classe também pode ser pensada como um **modelo** para os objetos que são instâncias da mesma. (O iPhone é uma classe. Até dezembro de 2010, as estimativas são de que 50 milhões de instâncias tinham sido vendidas!)

construtor
Cada classe tem uma "fábrica", **chamada** pelo mesmo nome da classe, por fazer novas instâncias. Se a classe tem um *método de **inicialização***, este método é usado para obter os atributos (ou seja, o estado) do novo objeto adequadamente configurado.

instância
Um objeto cujo tipo é de alguma classe. Instância e objeto são usados como **sinônimos**.

instanciar
Significa criar uma instância de uma classe e executar o seu método de inicialização.

linguagem orientada a objetos
Uma linguagem que fornece recursos, como as **classes** definidas pelo usuário e **herança**, que facilitam a programação orientada a objetos.

método
Uma **função** que é definida dentro de uma definição de classe e é chamado em instâncias dessa classe.

método de inicialização
Um método especial em Python (chamado **__init__**) que é chamado automaticamente para configurar um objeto recém-criado no seu estado inicial (padrão de fábrica).

objeto
Um tipo de **dados** composto que é frequentemente usado para modelar uma coisa ou conceito do mundo real.
Ele agrupa os **dados** e as **operações** que são relevantes para esse tipo de dados. Instância e objeto são usados como sinônimos.

programação orientada a objetos
Um estilo poderoso de programação em que os dados e as operações que os manipulam são organizados em classes e métodos.

- **Sintaxe**: POO em Python

```
class NomeClasse:                          # class NomeClasse (object):
    def __init__ (self, p_1, p_2, ...):    # Método construtor
        self.nome_atributo1 = p_1          # Atributos
        self.nome_atributo2 = p_2
        . . .
    def nome_metodo (self):                # Outros métodos da classe
        # script
        return

if __name__ == '__main__':                 # mai <tab>
    nome_objeto1 = NomeClasse(a_1, a_2, ...)  # Cria (instancia) o primeiro objeto da classe
    nome_objeto2 = NomeClasse(a_1, a_2, ...)  # Cria (instancia) o segundo objeto da classe
    . . .
    print ("Valor do atributo x: ", nome_objeto1.x)  # nome_objeto.nome_atributo
    nome_objeto1.y = yw2                   # alterar valor do atributo
    nome_objeto1.nome_metodo()             # Modelo pra chamar método da classe
    nome_objeto2.nome_metodo()             # Modelo pra chamar método da classe
```