Programação Orientada a Objetos para Dados

Classes e Objetos para Dados

Prof. Ms. Leonardo Rocha

- Unidade de Ensino: 2
- Competência da Unidade: Orientação a objetos em python.
- Resumo: Conhecer conceitos relativos à programação orientada a objetos em Python.
- Palavras-chave: OO, Python, pandas.s
- Título da Teleaula: Classes o objetos para dados
- Teleaula nº: 2

Contextualização

Conhecendo a Programação Orientada a Objetos

Ferramentas, Bibliotecas e habilidades técnicas

Estrutura de repetição e condição em Python

Biblioteca Pandas

Conceitos

Classes em Python

Classe

Classes proporcionam uma forma de organizar dados e funcionalidades juntos. Criar uma nova classe cria um novo "tipo" de objeto, permitindo que novas "instâncias" desse tipo sejam produzidas. Cada instância da classe pode ter atributos anexados a ela, para manter seu estado. Instâncias da classe também podem ter métodos (definidos pela classe) para modificar seu estado.

```
class Carro:                          #define a classe
        def __init__(self):           #método de inicialização
          print("carro pronto!")      #escreve mensagem
```

Fonte: https://docs.python.org/pt-br/3/tutorial/classes.html

Objetos

A regrinha de criação de objetos é simples. Uma vez que exista uma classe, por exemplo, chamada carro, nós podemos criar nossos objetos instanciando a classe. Veja:

```
class Carro:                          #define a classe
    tipo = "sedan"
        def __init__(self):           #método de inicialização
            print("carro pronto!")    #escreve mensagem
corolla = Carro()
civic   = Carro()
```

Atributos

Atributo é herdado ao instanciar uma classe a um objeto. Agora, é possível imprimir o atributo do novo objeto, dessa forma:

```
>>>print(corolla.tipo)
```

Exemplo

```
class Carteira:
    pass


carteira_1 = Carteira()
    print(carteira_1)
<__main__.Carteira object at 0x000001B755875700>
carteira_2 = Carteira()
    print(carteira_2
<__main__.Carteira object at 0x000001B7558758B0>
```

Atributos e Métodos

```
In [1]:
1 class Carteira:
2     saldo = 0
```

podemos acessar os atributos e métodos do nosso objeto usando a sintaxe

<objeto>.<atributo ou método>

```
>>> c = Carteira()
>>> print(c.saldo)
0
```

Conceitos

Parâmetro Especial

Parâmetro especial

quase todos os métodos vão exigir um parâmetro (argumento) especial, conhecido por **self** e é obrigatório como primeiro parâmetro pois, através dele, acessaremos os atributos e métodos da classe.

Exemplo

In [2]:
```
class Carteira:
    saldo = 0

    def adicionar_fundos(self, valor):
        self.saldo += valor
        print('Operação realizada com sucesso!')

    def remover_fundos(self, valor):
        if self.saldo >= valor:
            self.saldo -= valor
            print('Operação realizada com sucesso!')
        else:
            print('Operação não realizada. Saldo insuficiente.')
```

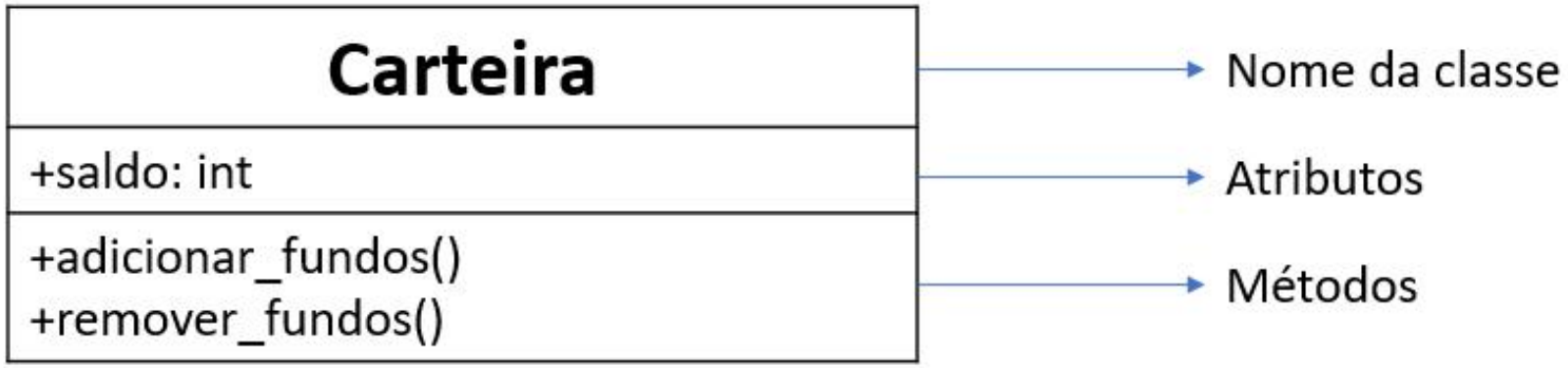
Agora podemos manipular nossa classe e, com o auxílio dos novos métodos, manipular nosso saldo.

Parâmetro especial

```
>>> c = Carteira()
>>> print(c.saldo)
0
>>> c.adicionar_fundos(50)
Operação realizada com sucesso!
>>> print(c.saldo)
50
>>> c.remover_fundos(30)
Operação realizada com sucesso!
```

UML

Linguagem de Modelagem Unificada contém notação gráfica para documentar um projeto. Conta com artefatos para classes, objetos e pacotes que representam programas OO. Diagrama de classe é um exemplo:



Resolução da SP

Como criar duas novas classes para cash back (cartão gold e platinum) de 3% e 5%?

Código

In [2]:

```
class CartaoGold(CartaoBase):
    def calcular_cash_back(self, valor):
        return valor * 0.03


class CartaoPlatinum(CartaoBase):
    def calcular_cash_back(self, valor):
        return valor * 0.05

```

```
print(CartaoGold().calcular_cash_back(100))
print(CartaoPlatinum().calcular_cash_back(100))
```

Interação

Dúvidas

Conceitos

OO com numPy

NumPy

É uma biblioteca open source para trabalhar com computação científica, pois oferece uma poderosa estrutura de dados multidimensional que nos permite realizar operações de forma eficiente e rápida, incluindo matemática, lógica, classificação, álgebra linear básica, estatística básica, dentre outras

Exemplo

```
>>> import numpy as np
>>> minha_lista=[10, 20, 30]
>>> meu_array=np.array(minha_lista)
>>> print(meu_array)
[10 20 30]
>>> print(type(meu_array))
<class 'numpy.ndarray'>
>>> meu_array.size
3
>>> meu_array.max()
30
```

Atributos e métodos do objeto ndarray está disponível:

https://bit.ly/3oujEqQ

Documentar o código

Docstrings deve ser inserido nas classes para orientar sobre o código ou até mesmo o uso por outro programa.

Pode ser escrito com aspas simples ('), duplas (") ou triplas (""") e devem conter uma descrição do propósito da classe ou método.

Classe carteira

In [1]:

```python
class Carteira:
    "Representa uma carteira digital para armazenar a manipular um saldo monetário."
    saldo = 0

    def adicionar_fundos(self, valor):
        "Adiciona o valor indicado no saldo da carteira."
        self.saldo+= valor
        print(f'Valor adicionado com sucesso! Seu novo saldo é de R${self.saldo}.')

    def remover_fundos(self, valor):
        "Remove o valor indicado da carteira, caso o saldo seja suficiente."
        if self.saldo >= valor:
            self.saldo -= valor
            print(f'Operação realizada com sucesso!')
        else:
            print('Operação não realizada. Saldo insuficiente.')
```

Exemplo de uso no Pandas

```
@final
def head(self: FrameOrSeries, n: int = 5) -> FrameOrSeries:
    """
    Return the first `n` rows.

    This function returns the first `n` rows for the object based
    on position. It is useful for quickly testing if your object
    has the right type of data in it.

    For negative values of `n`, this function returns all rows except
    the last `n` rows, equivalent to ``df[:-n]``.

    Parameters
    ----------
    n : int, default 5
        Number of rows to select.

    Returns
    -------
    same type as caller
        The first `n` rows of the caller object.

    See Also
    --------
    DataFrame.tail: Returns the last `n` rows.

    Examples
    --------
```

Classe abstrata

define os métodos e atributos que suas classes filhas terão, estas, no entanto, podem ter outros atributos e métodos próprios, além dos que foram previamente definidos em sua classe pai (LOTT, 2019). Classes derivadas das classes abstradas são chamadas de classes concretas.

Criar classes abstradas exige a utilização do módulo **abc** que oferece a infraestrutuar necessária para que os mecanismos de heranaça e abstração funcionem como o esperado.

Exemplo

In [1]:

```python
from abc import ABC, abstractmethod


class CartaoBase(ABC):
    """
    Classe abstrata que representa um cartão de crédito
    """
    @abstractmethod
    def calcular_cash_back(self, valor):
        pass
```

Conceitos

Métodos e Herança

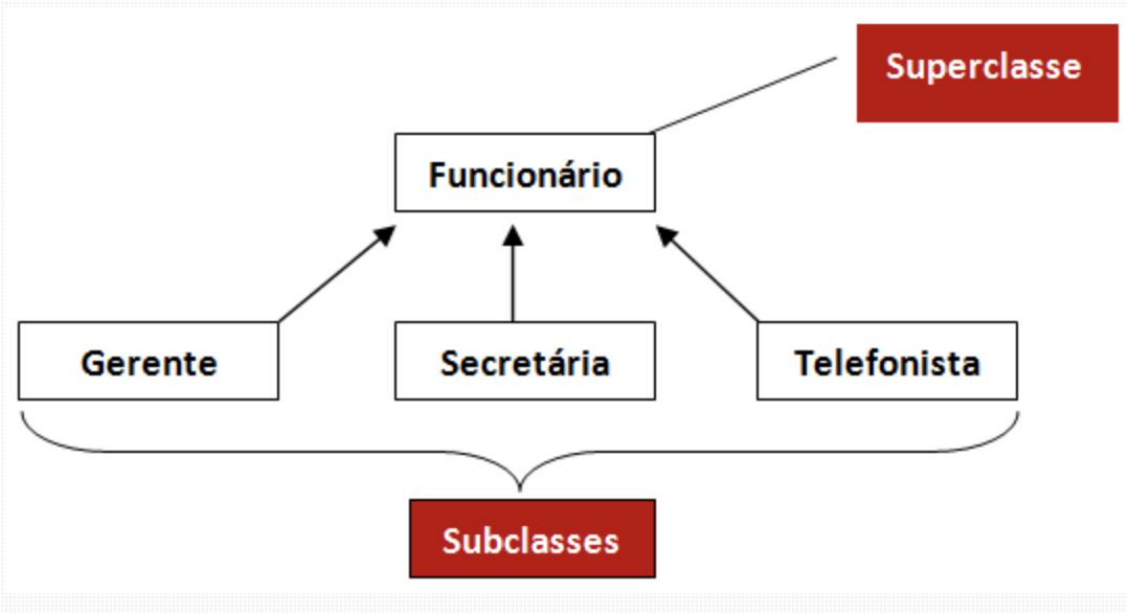
Métodos

Quando queremos criar um objeto com um “estado” já pré-estabelecido, em Pyhton, pode-se definir um método especial chamado __init__(), na classe. Esse método construtor inicia objetos com atributos padrão que precisam obrigatoriamente existir em um objeto de forma válida, ou que necessitam ser iniciados primeiro na execução do código (PHILLIPS, 2018).

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6  x = Point(1, 2)
7  print(x.x)
```

Herança

Ela nos permite criar relacionamentos entre duas ou mais classes, abstraindo a lógica comum em superclasses e gerenciando detalhes específicos na subclasse.” (PHILLIPS, 2018, p.67).



Na prática

In [13]:
```python
class Conta:
    def __init__(self, numero_da_conta, saldo):
        self.numero_da_conta = numero_da_conta
        self.saldo = saldo

    def depositar(self, valor):
        self.saldo += valor

    def sacar(self, valor):
        self.saldo -= valor
```

In [14]:
```python
class ContaPoupanca(Conta):
    def calcular_rendimentos(self):
        "Adiciona 0,05% ao saldo da conta."
        self.saldo += self.saldo * 0.005
```

Ao criar um novo objeto usando a classe ContaPoupanca, podemos usar os métodos definidos na classe pai Conta e o método que foi definido na própria classe filha

Na prática

```
In [15]:
1 cp = ContaPoupanca(numero_da_conta=123456, saldo=200)
```

```
In [16]:
1 cp.depositar(100)
2 cp.saldo
```

```
Out[16]: 300
```

```
In [17]:
1 cp.sacar(50)
2 cp.saldo
```

```
Out[17]: 250
```

```
In [18]:
1 cp.calcular_rendimentos()
2 cp.saldo
```

```
Out[18]: 251.25
```

Resolução da SP

Como instanciar uma classe que contenha um método abstrato?

Solução

In [34]:
```
from abc import ABC, abstractmethod
```

In [35]:
```
class Conta(ABC):
    def __init__(self, numero_da_conta, saldo):
        self.numero_da_conta = numero_da_conta
        self.saldo = saldo

    def depositar(self, valor):
        self.saldo += valor

    @abstractmethod
    def sacar(self, valor):
        raise NotImplementedError
```

Sobrescrever método sacar

In [41]:
```
class ContaPoupanca(Conta):
    def sacar(self, valor):
        if valor <= self.saldo:
            self.saldo -= valor
        else:
            print('A operação não pode ser realizada.')

    def calcular_rendimentos(self):
        "Adiciona 0,05% ao saldo da conta."
        self.saldo += self.saldo * 0.005
```

Interação

Dúvidas

Conceitos

Recapitulando

Orientação a objetos em Python

Criação de classes

Instanciar classes

Biblioteca Pandas e NumPy