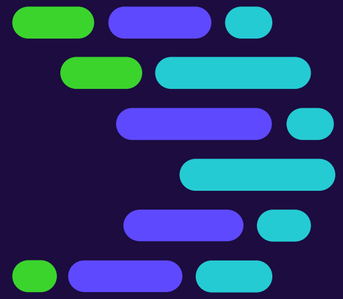


devaria

Programação – Python

Parte 3

Parte 1

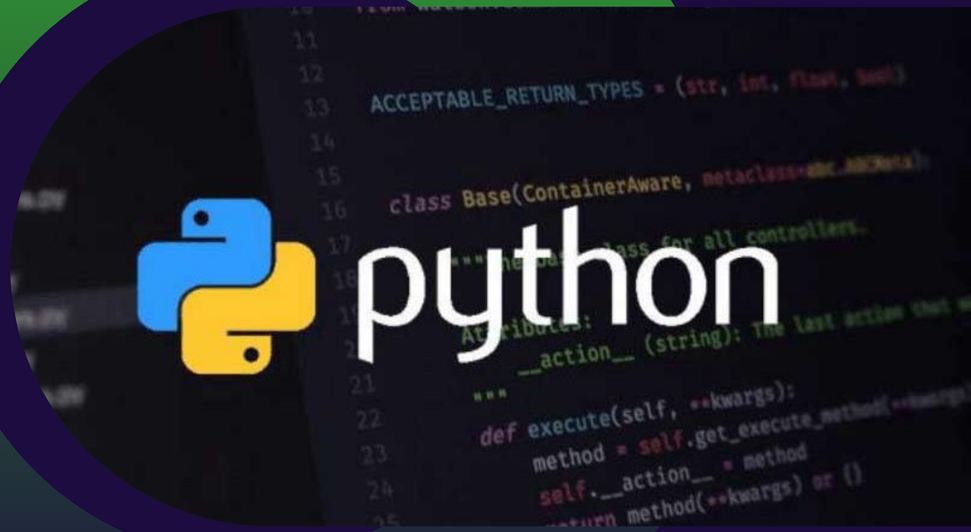
- Resumo
- Desafio Classe

Estrutura de repetição

```
for i in range(0, 10):
        print(i) # 0 1 2 3 4 5 6 7 8 9
```

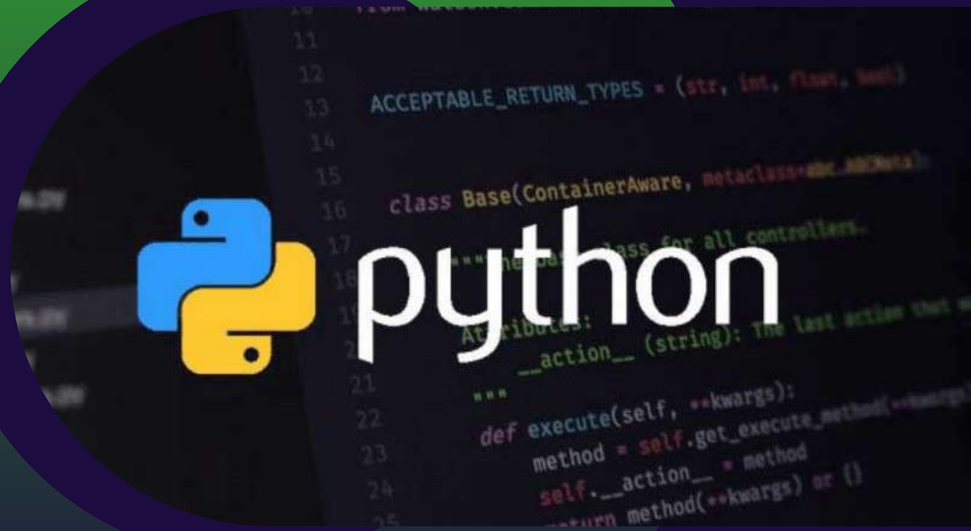
```
palavra = 'devaria'
for letra in palavra:
        print(p) # d e v a r i a
```

```
i = 0
while i < 10:
        print(i)  # 0 1 2 3 4 5 6 7 8 9
        i += 1
```



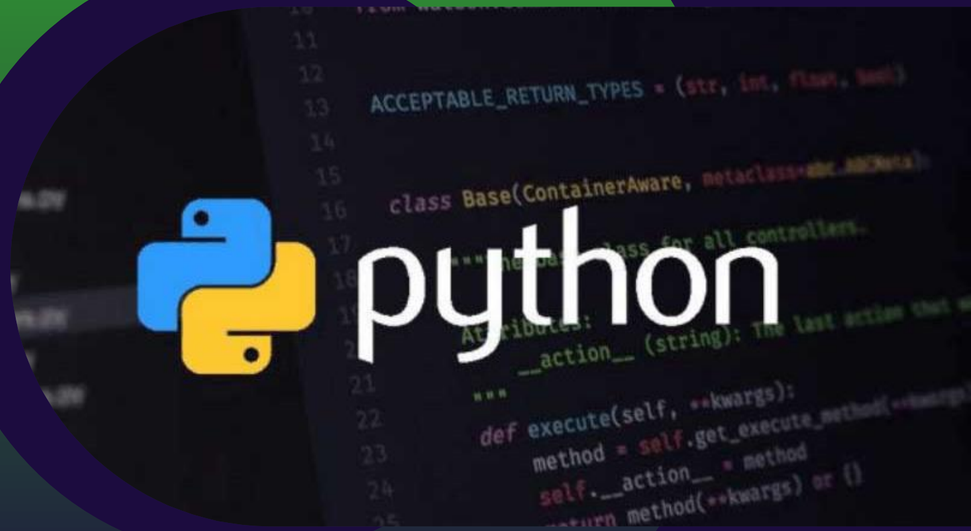
Try/Except/Finally

```
mensagem = “”
try:
        n1 = int(input(“Digite o primeiro número: ”))
        n1 = int(input(“Digite o primeiro número: ”))
        n1 = int(input(“Digite o primeiro número: ”))

        mensagem = “Todos os inteiros lidos
                com sucesso.”
except Exception as e:
        mensagem = “Você informou um número inválido”
finally:
        print(mensagem)
```



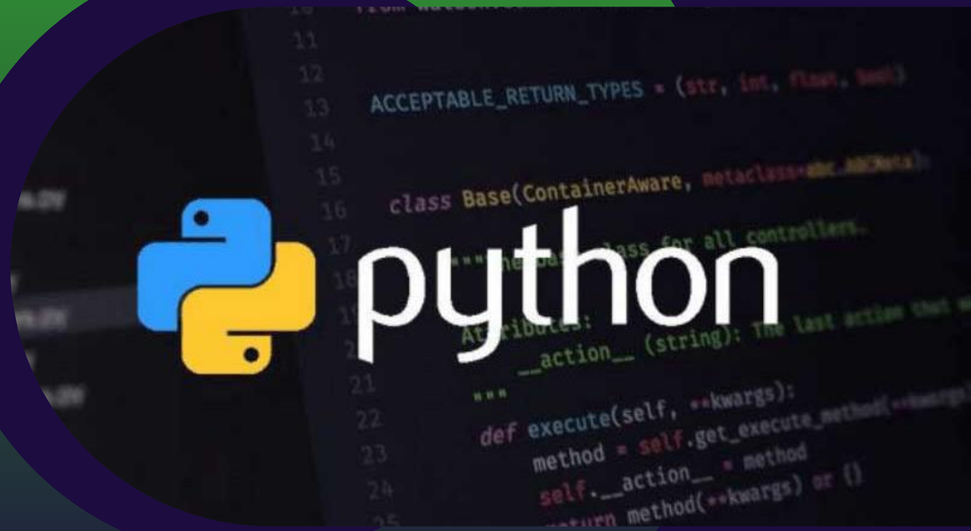
Funções anônimas

```
# Filtrar notas acima de 80.

notas_alunos = [ 90, 71, 82, 93, 75, 82 ]

notas_filtradas = list(filter(lambda i : i > 80, notas_alunos))

print(f'Notas acimas de 80 {notas_filtradas}')
```

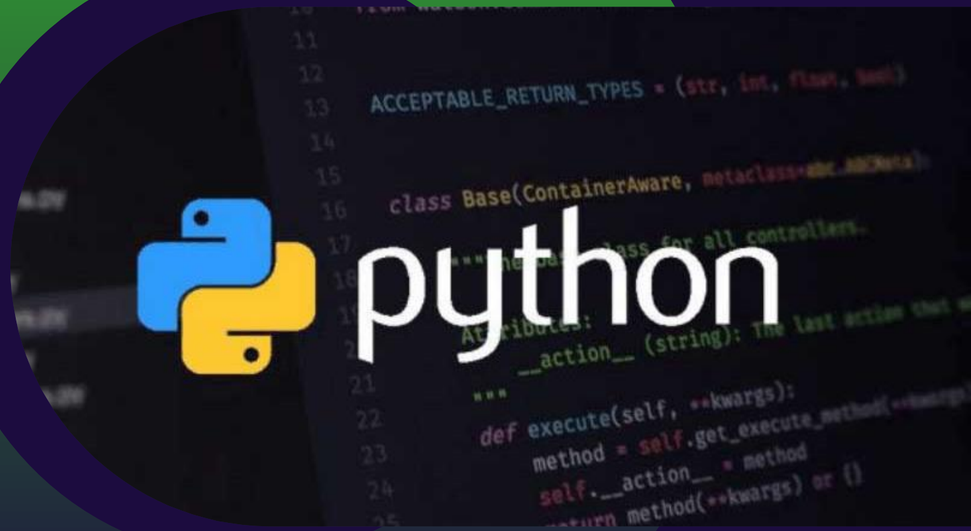


Classes

```
class ContaBancaria:
        def __init__(self, tipo, numero, agencia):
                self.tipo = tipo
                self.numero = numero
                self.agencia = agencia


        def exibirDadosConta(self):
                print(”Conta, tipo: {self.tipo},
            agencia: {self.agencia} e numero: {self.numero}”)
```



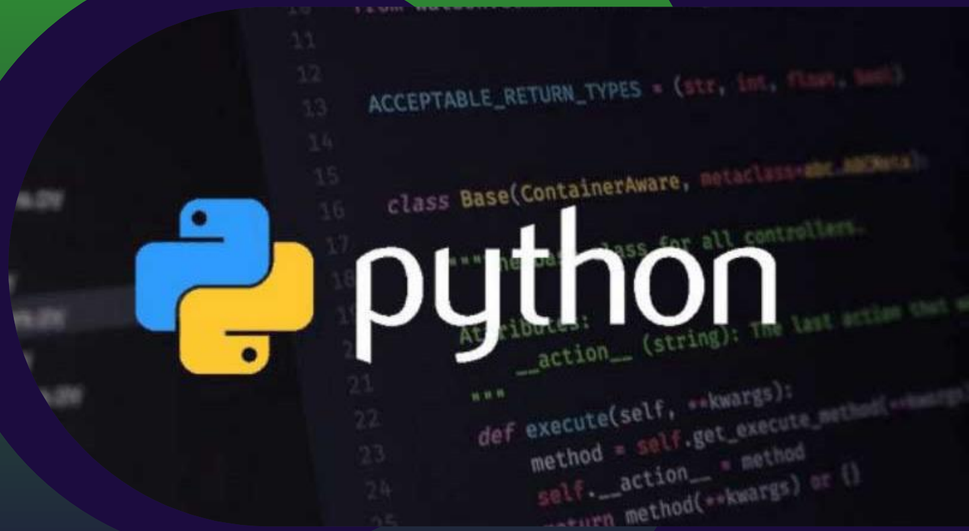
Hora da prática



Agora que já entendemos como funcionam classes, objetos, construtores, atributos e métodos. Nesse exercício vamos exercitar algumas coisas:

- Criar um novo programa;
- Criar uma classe;
- Transformar nossos argumentos em objetos;
- Retornar para o usuário o resultado do programa;

Hora da prática



Desafio (só que agora com classes =D):

Escrever um programa que recebe alguns produtos como argumento, validar se esse produtos estão na lista de itens disponíveis do mercado, caso esteja avisar o usuário quais produtos tem e quais não tem e por ultimo exibir a lista de produtos disponíveis ordenados por ordem alfabética do mercado para que o usuário possa pedir na próxima vez.

Café ou Dúvidas?

Para que a aula não fique cansativa, a cada conclusão de assunto teremos 15 minutos para que vocês possam tirar as dúvidas através dos comentários no vídeo e responderemos ao vivo.

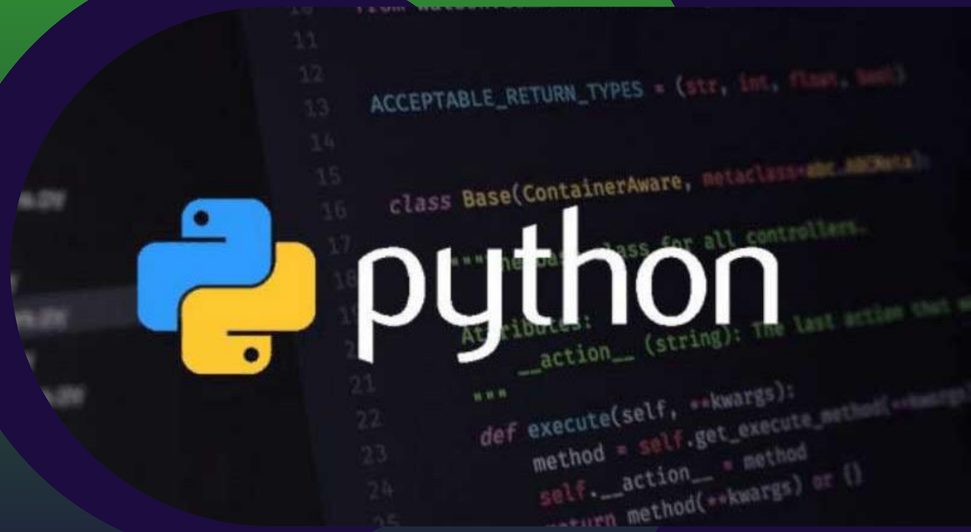
Parte 2

- Encapsulamento
- Herança

Encapsulamento

O “Encapsulamento” vem de encapsular, que em programação orientada a objetos significa separar o programa em partes, as mais isoladas possíveis. A ideia é tornar o software mais flexível, fácil de modificar e de criar novas implementações”.

A classe bem encapsulada deve ocultar seus dados e os detalhes de implementação do mundo exterior. Isso é denominado programação caixa preta. Usando o encapsulamento, a implementação do método pode ser alterada pelo autor da classe sem quebrar qualquer código existente fazendo uso dela.



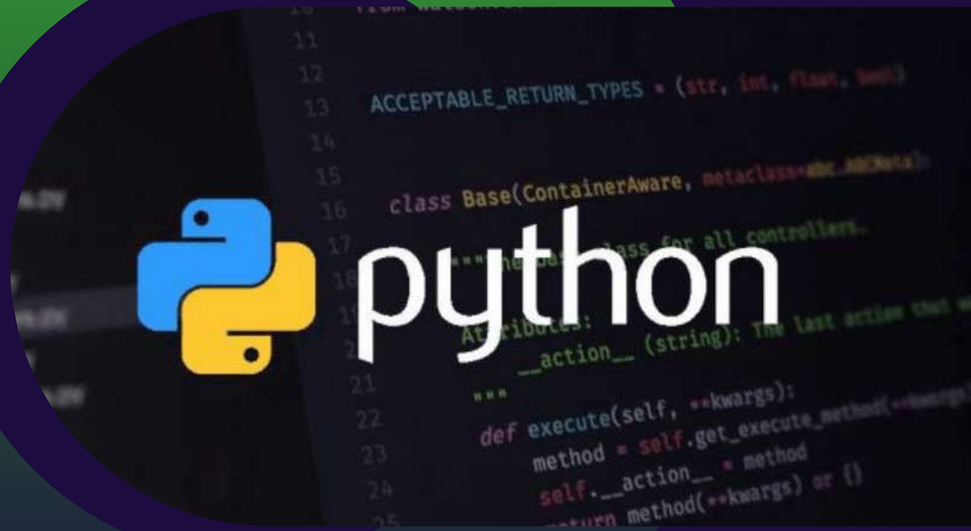
Encapsulamento

Um modificador de acesso define o escopo e a visibilidade de um atributo ou método da classe.

A linguagem Python suporta os seguintes modificadores de acesso:

- Public - Qualquer atributo/método público pode ser acessado de fora da classe através do operador . no objeto instanciado;

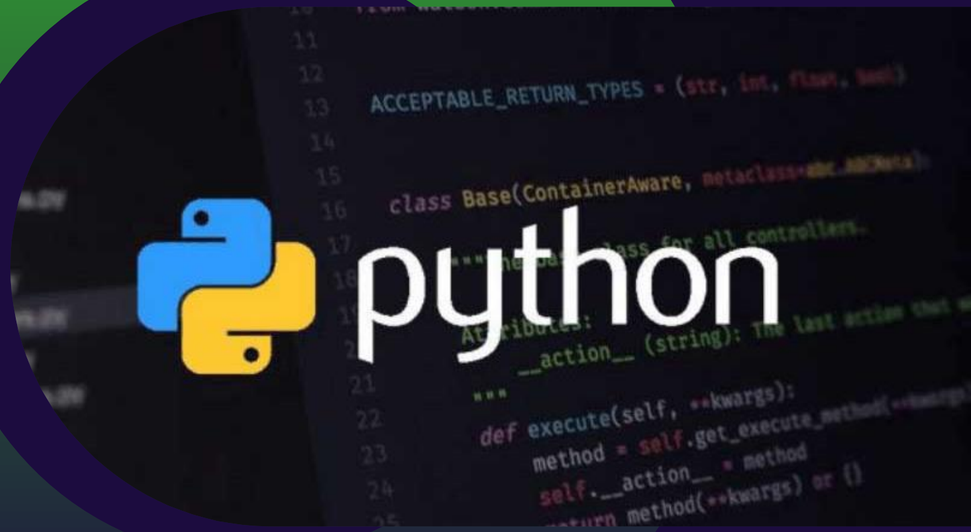
- Private - Qualquer atributo/método privado é de uso interno e exclusivo da classe, só podendo ser acessado por outros atributos/métodos e construtores;



Encapsulamento

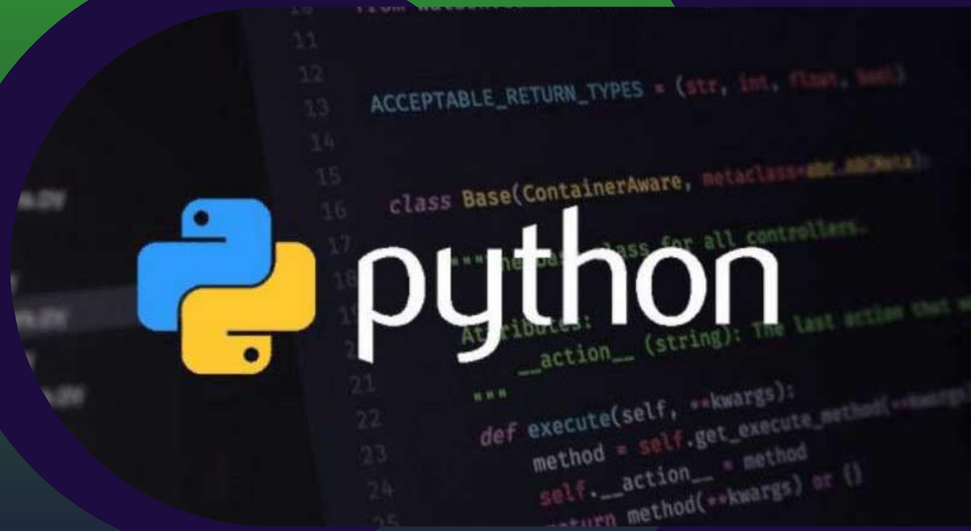
No Python por padrão todo método e atributos de classes são públicos.

Para definir um atributo ou método privado devemos iniciar seu nome com dois underscores “__”



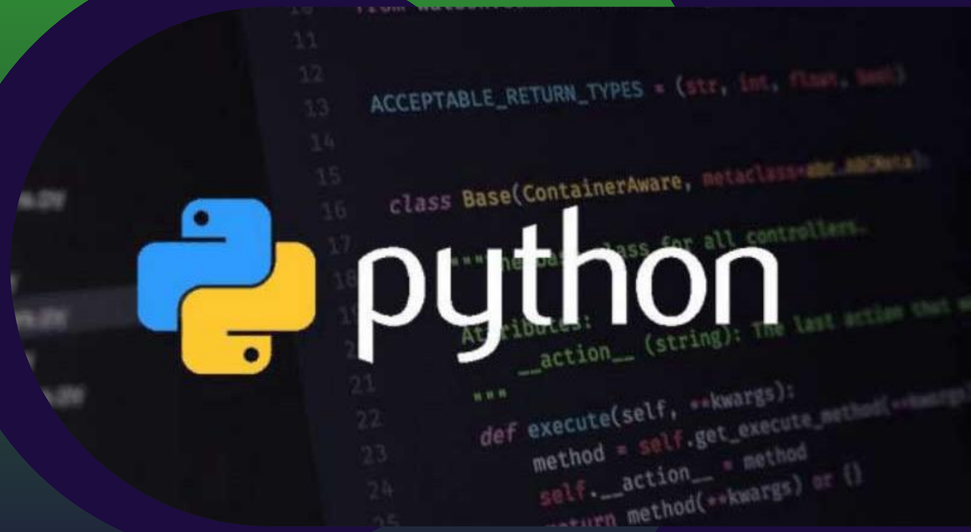
Encapsulamento

```
class ContaBancaria:
        def __init__(self, tipo, numero, agencia, saldo):
                self.tipo = tipo
                self.numero = numero
                self.agencia = agencia
                self.__saldo = saldo

        def exibirDadosConta(self):
                print(”Conta, tipo: {self.tipo},
            agencia: {self.agencia} e numero: {self.numero}”)


        def exibirSaldo(self):
                print(”Seu saldo é: {self.__saldo}”)
```



Encapsulamento

```
conta1 = ContaBancaria(“Corrente", 35662, "0001“, 0)

conta1.tipo # Funciona porque o atributo é publico

conta1.numero = 234123 # Funciona porque o atributo é publico

conta1.__saldo # Não funciona pegar o valor porque é privado

conta1.exibirSaldo() # Funciona porque o método é publico
```



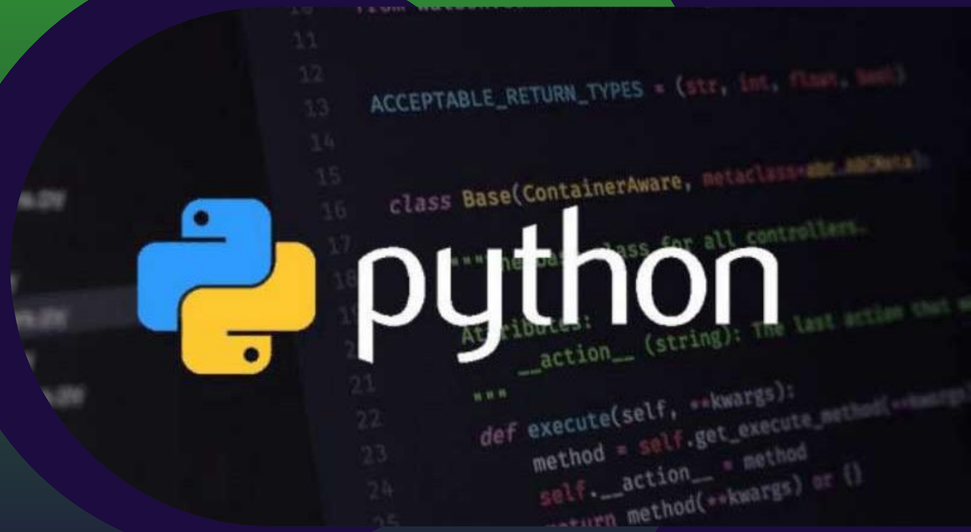
Herança

A herança, assim como o encapsulamento e o polimorfismo, é uma das três principais características da programação orientada ao objeto.

A herança permite que você crie novas classes que reutilizam, estendem e modificam o comportamento definido em outras classes.

A classe cujos membros são herdados é chamada classe base ou classe pai e a classe que herda esses membros é chamada classe derivada ou classe filha.

A herança é transitiva. Se ClassC é derivado de, ClassB e ClassB é derivado de ClassA , ClassC herda os membros declarados em ClassB e ClassA .



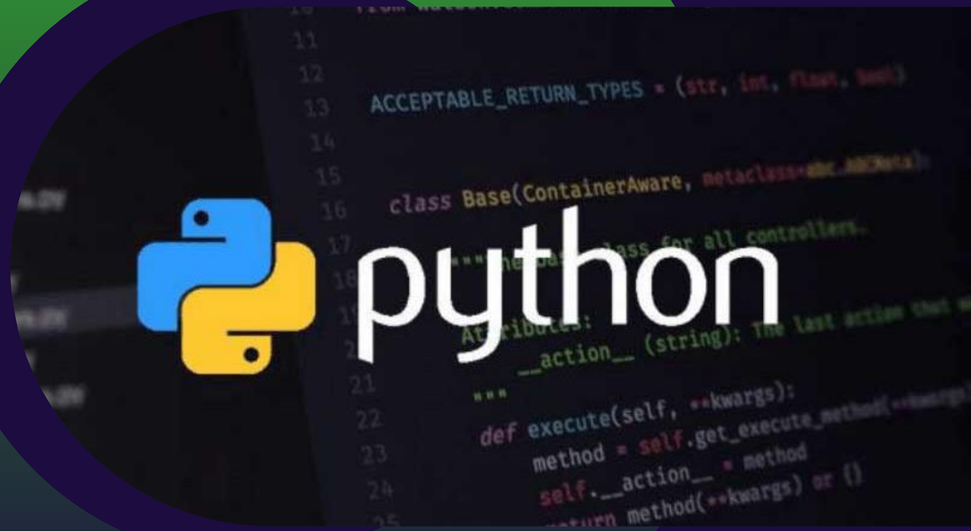
Herança

Conceitualmente, uma classe derivada é uma especialização da classe base.

Por exemplo, se tiver uma classe base Animal, você pode ter uma classe derivada chamada Mamífero e outra classe derivada chamada Réptil.

Um Mamífero é um Animal e um Réptil é um Animal, mas cada classe derivada representa especializações diferentes da classe base.

E nossa missão ao utilizar herança é separar aquilo que é comum para a classe base e as especializações de cada variedade na classe filha

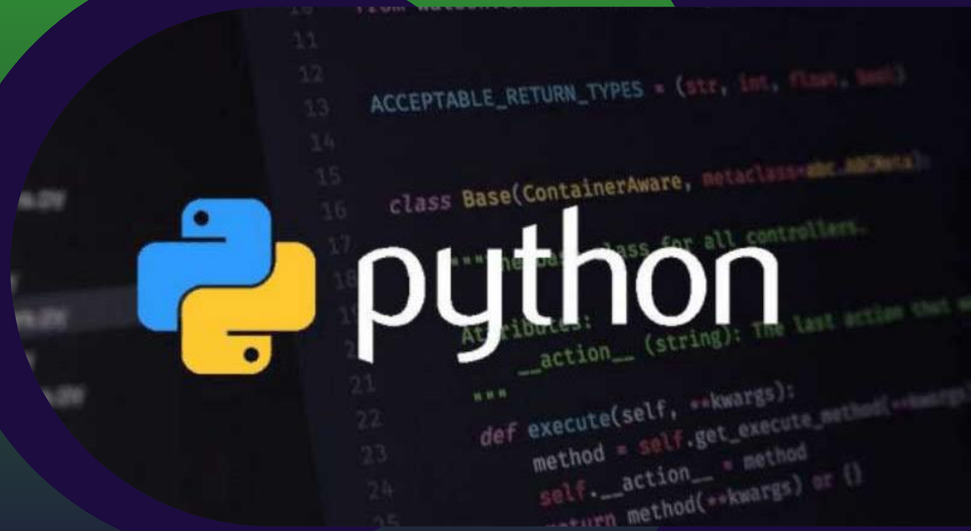


Herança

```
class Animal:
    def __init__(self, nome):
        self.nome = nome

    def Respirar(self):
        print("O animal {self.nome} está respirando")
```

```
class Mamifero(Animal):
    def __init__(self, nome):
        super().__init__(nome)

    def Mamar(self):
        print("O animal {self.nome} está se alimentando")
```

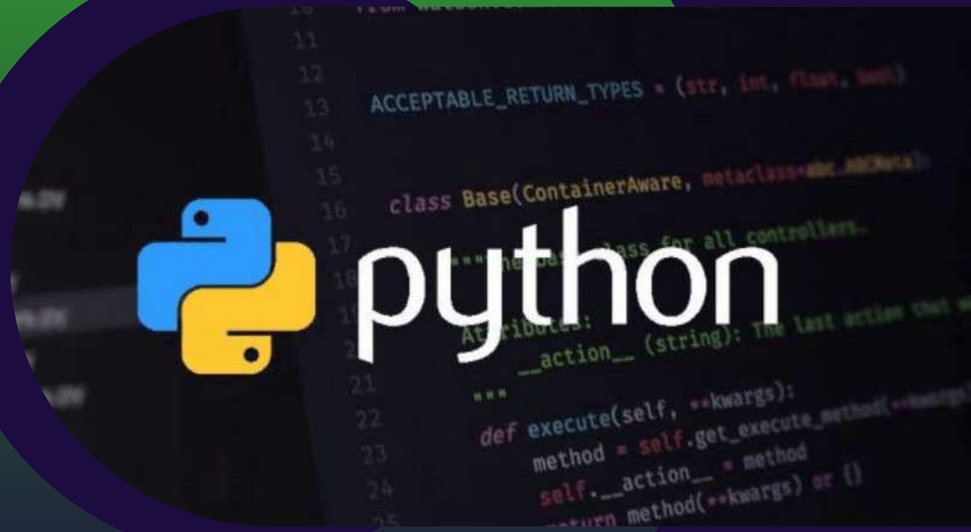
```
class Reptil(Animal):
    def __init__(self, nome):
        super().__init__(nome)

    def BotarOvo(self):
        print("O animal {self.nome} está botando ovo")
```



Hora da prática

Agora que já entendemos como funcionam exceções, listas e funções anônimas. Nesse exercício vamos exercitar algumas coisas:

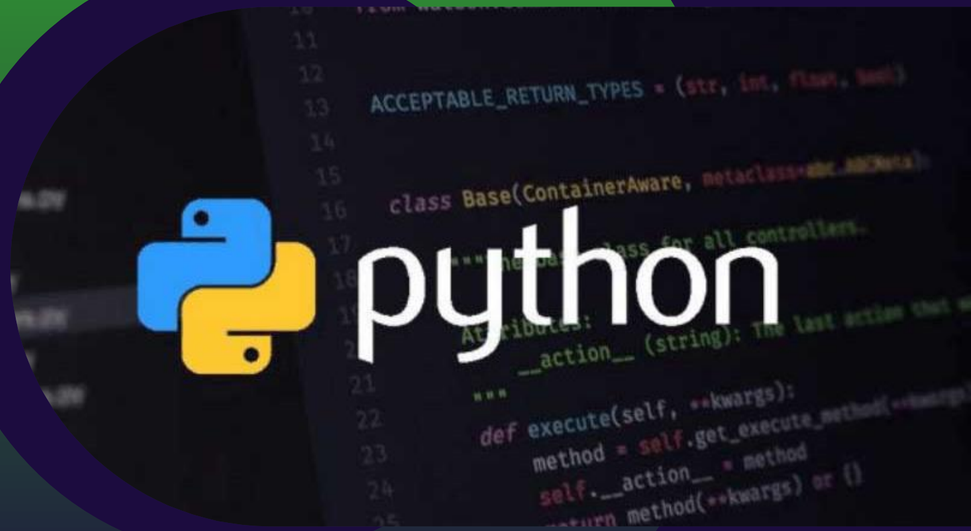
- Criar um novo programa;
- Utilizar encapsulamento nas classes;
- Utilizar herança nas classes;
- Retornar para o usuário o resultado do programa;



Hora da prática

Desafio:

Escrever um programa que recebe um nome de animal vertebrado e de acordo com um filtro identifica qual dos grupos ele pertence (Mamífero, Réptil, Ave e Peixe) e exibir os dados genéricos e exclusivos de cada grupo de animal



Café ou Dúvidas?

Para que a aula não fique cansativa, a cada conclusão de assunto teremos 15 minutos para que vocês possam tirar as dúvidas através dos comentários no vídeo e responderemos ao vivo.

Parte 3

- Polimorfismo
- Enums
- Interfaces/Implementações

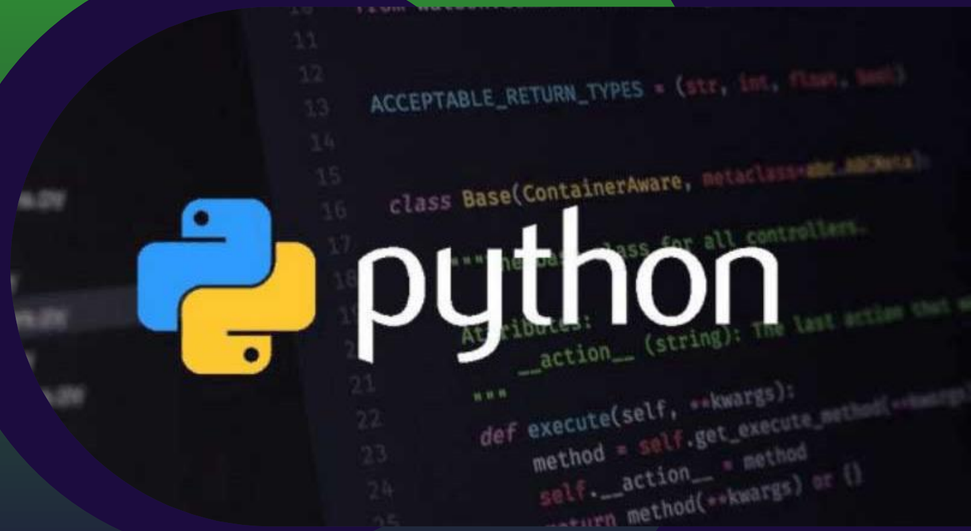
Polimorfismo

O polimorfismo costuma ser chamado de o terceiro pilar da programação orientada a objetos, depois do encapsulamento e a herança.

O polimorfismo é uma palavra grega que significa "de muitas formas" e tem dois aspectos distintos:

- Em tempo de execução, os objetos de uma classe derivada podem ser tratados como objetos de uma classe base, em locais como parâmetros de método, coleções e matrizes. Quando este polimorfismo ocorre, o tipo declarado do objeto não é mais idêntico ao seu tipo de tempo de execução.

- As classes base podem definir e implementar métodos virtuais e as classes derivadas podem substituí-los, o que significa que elas fornecem sua própria definição e implementação. Em tempo de execução, quando o código do cliente chama o método, o CLR procura o tipo de tempo de execução do objeto e invoca a substituição do método virtual. No código-fonte, você pode chamar um método em uma classe base e fazer com que a versão de uma classe derivada do método seja executada.

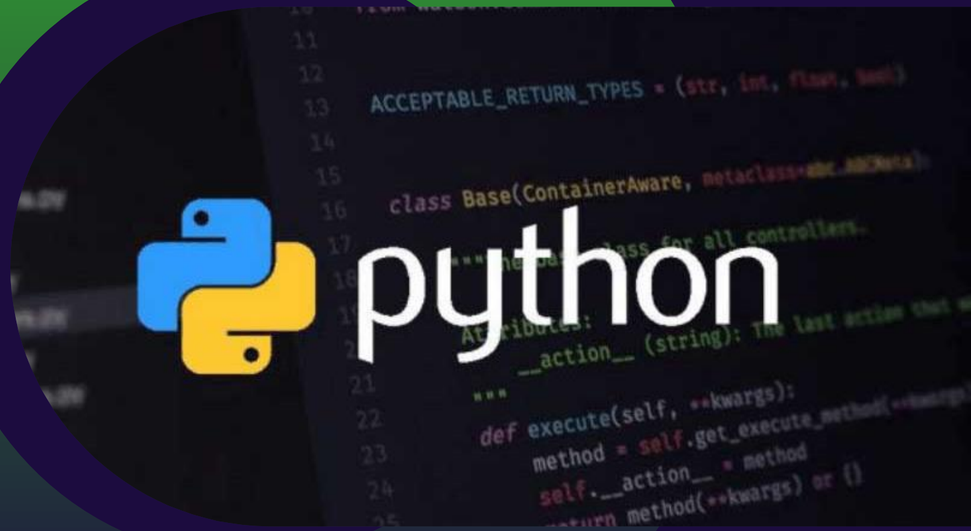


Polimorfismo

```
class FormaGeometrica:
    def __init__(self):
        pass

    def Desenhar(self):
        print("Você está desenhando uma forma geométrica.")
```

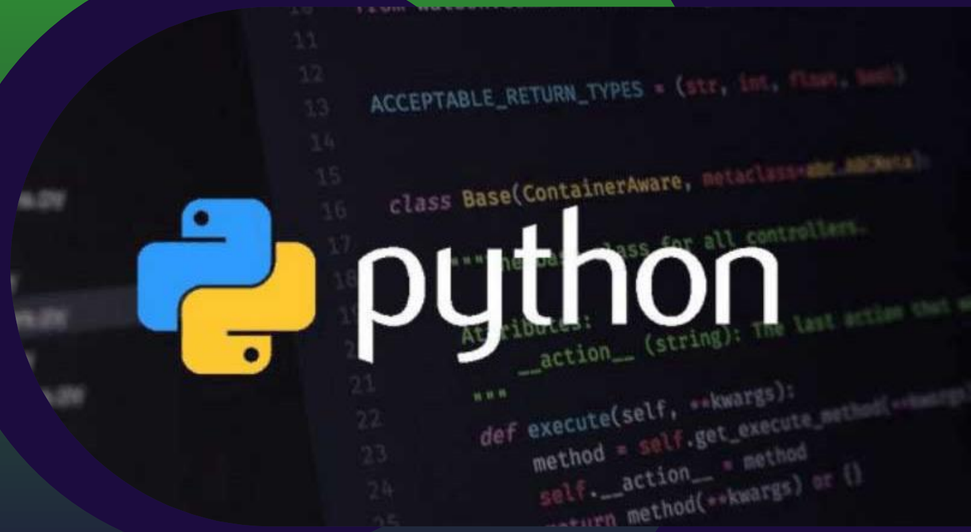
```
class Circulo(FormaGeometrica):
    def __init__(self):
        pass

    def Desenhar(self):
        print("Você está desenhando um Circulo")
```

```
class Quadrado(FormaGeometrica):
    def __init__(self):
        pass

    def Desenhar(self):
        print("Você está desenhando um Quadrado")
```



Polimorfismo

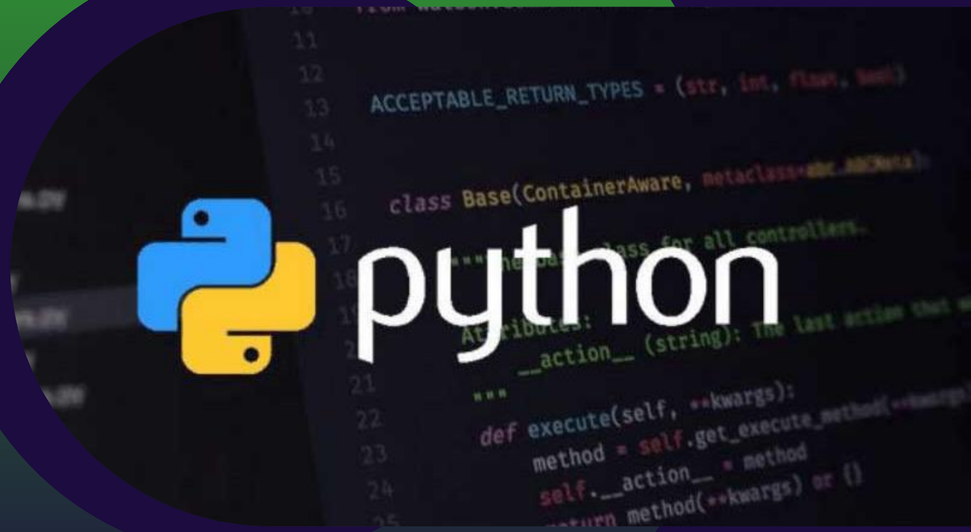
```
formas = [
        Quadrado(),
        Circulo()
]

for forma in formas:
        forma.Desenhar()
```



Enums

Um tipo de enumeração (ou enum) é um tipo de valor definido por um conjunto de constantes nomeadas do tipo numérico integral subjacente. Para definir um tipo de enumeração, use a enum palavra-chave e especifique os nomes dos membros de enumeração:

```
class EstacoesDoAno(Enum):
        VERAO = 'Verão'
        OUTONO = 'Outono'
        INVERNO = 'Inverno'
        PRIMAVERA = 'Primavera'


class CodigosDeErro(Enum):
        Vazio = 0
        Desconhecido = 1
        UsuarioInvalido = 2
        DadosInvalidos = 3
```

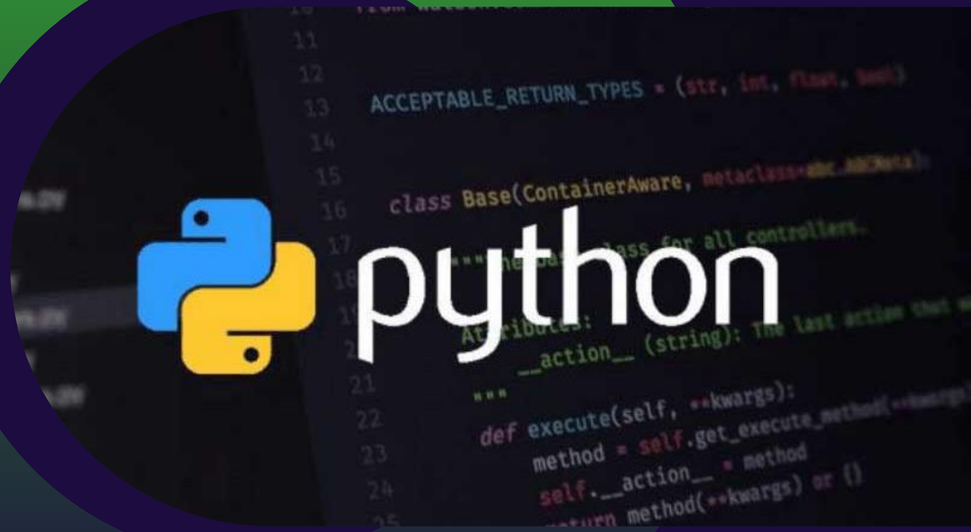


Interfaces/Implementações

Um outro tipo de arquivo muito importante na orientação a objetos é a interface. Este tipo de arquivos costumamos de chamar de contrato. São uma lista de métodos/atributos que qualquer classe que implementar “assinar” esse contrato deve obedecer.

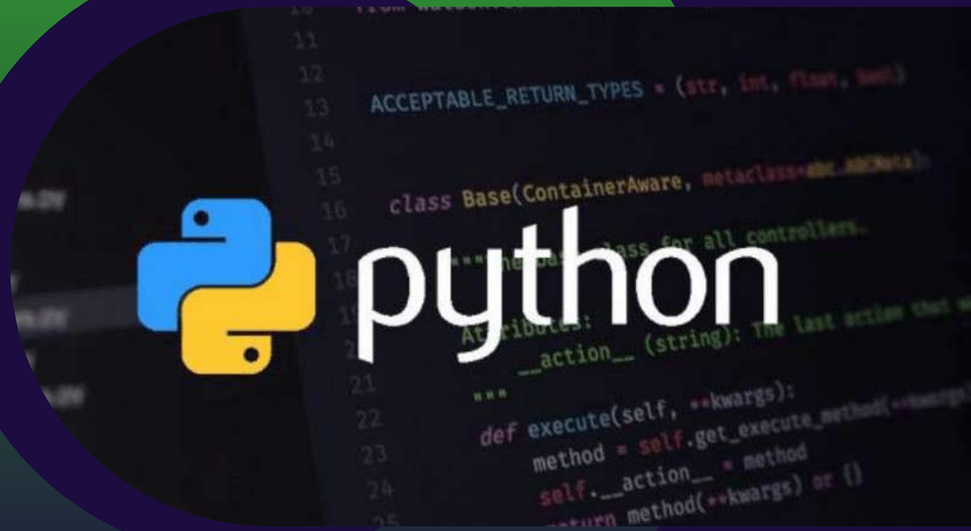
Utilizamos interfaces para definir comportamentos padrões que serão implementados classe a classe muito parecido com polimorfismo, porém não tem um vinculo com herança, seria como várias classes assinando um contrato de comportamentos que aquela classe deve ter

Ao contrario da herança, uma classe pode assinar quantos contratos ela quiser e caso algum comportamento não seja definido o compilador nem deixa seguir com o programa.



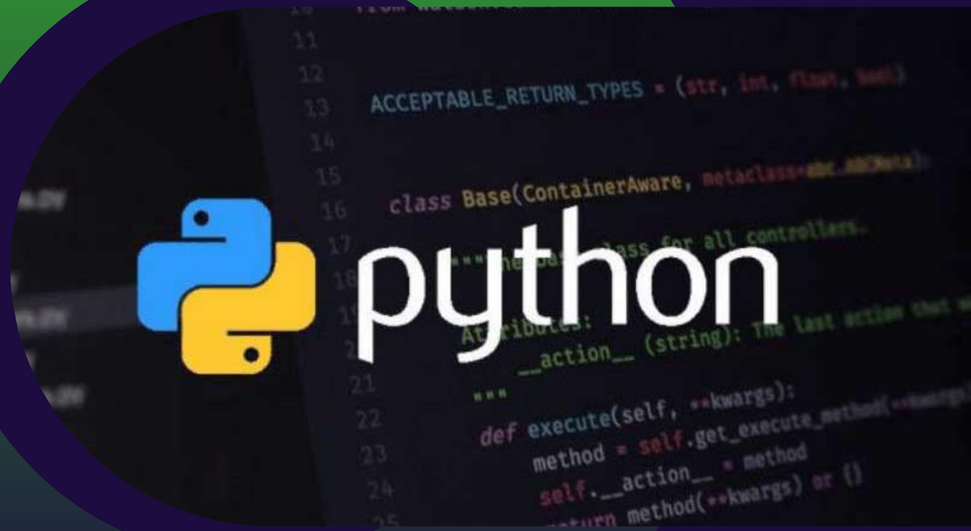
Interfaces/Implementações

```
import abc

class Formulario (abc.ABC):

        @abc.abstractmethod
        def validarCampos(self):
                pass

class Impressao (abc.ABC):

        @abc.abstractmethod
         def imprimirComprovante(self, produtos):
                pass
```

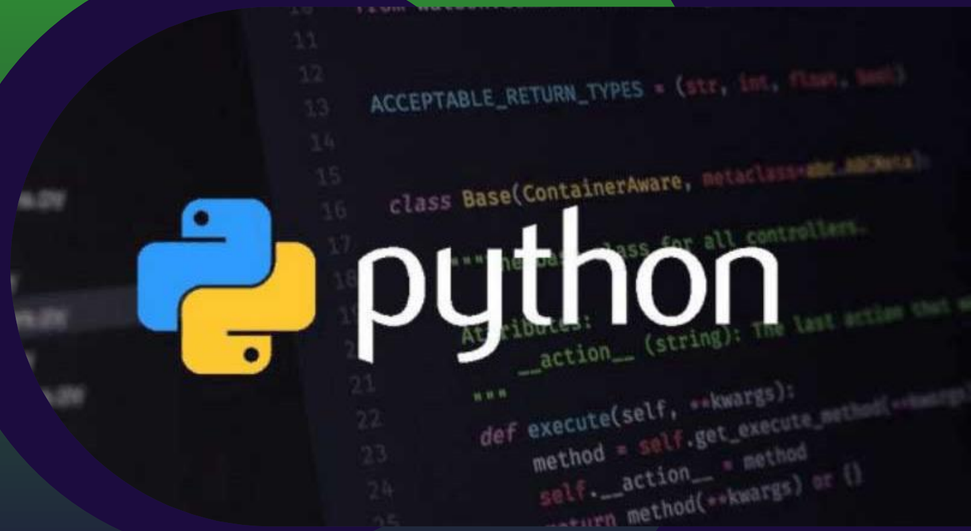


Interfaces/Implementações

```
classe ContaBancariaFormulario (Formulario,
Impressao):
        def validarCampos():
            print(f”Campos validados com
sucesso”)

            return True

        def imprimirComprovante(produtos):
            print(f”Imprimindo comprovante dos
            produtos”)
```



Hora da prática



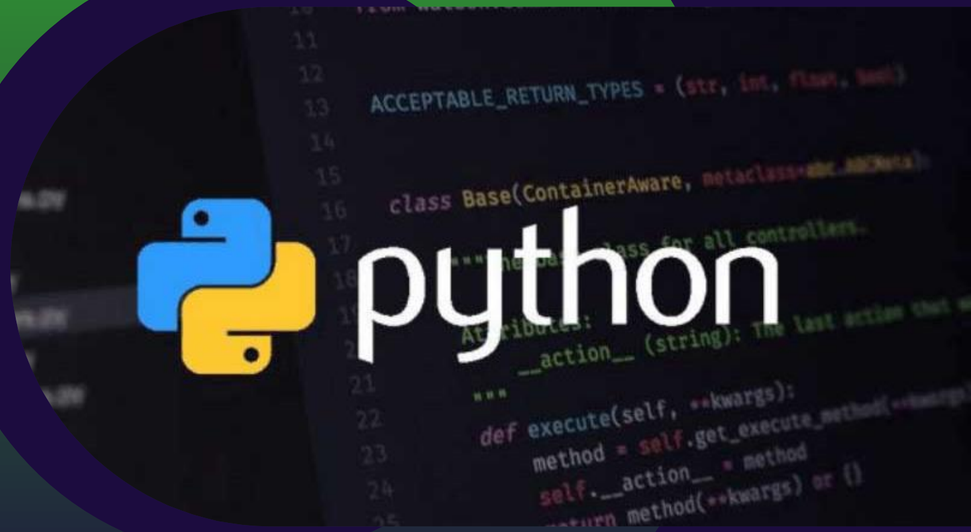
Agora que já entendemos como funcionam classes, objetos, construtores, atributos e métodos. Nesse exercício vamos exercitar algumas coisas:

- Criar um novo programa;
- Criar uma classes e comportamentos;
- Utilizar de polimorfismo;
- Criar enumerações;
- Retornar para o usuário o resultado do programa;

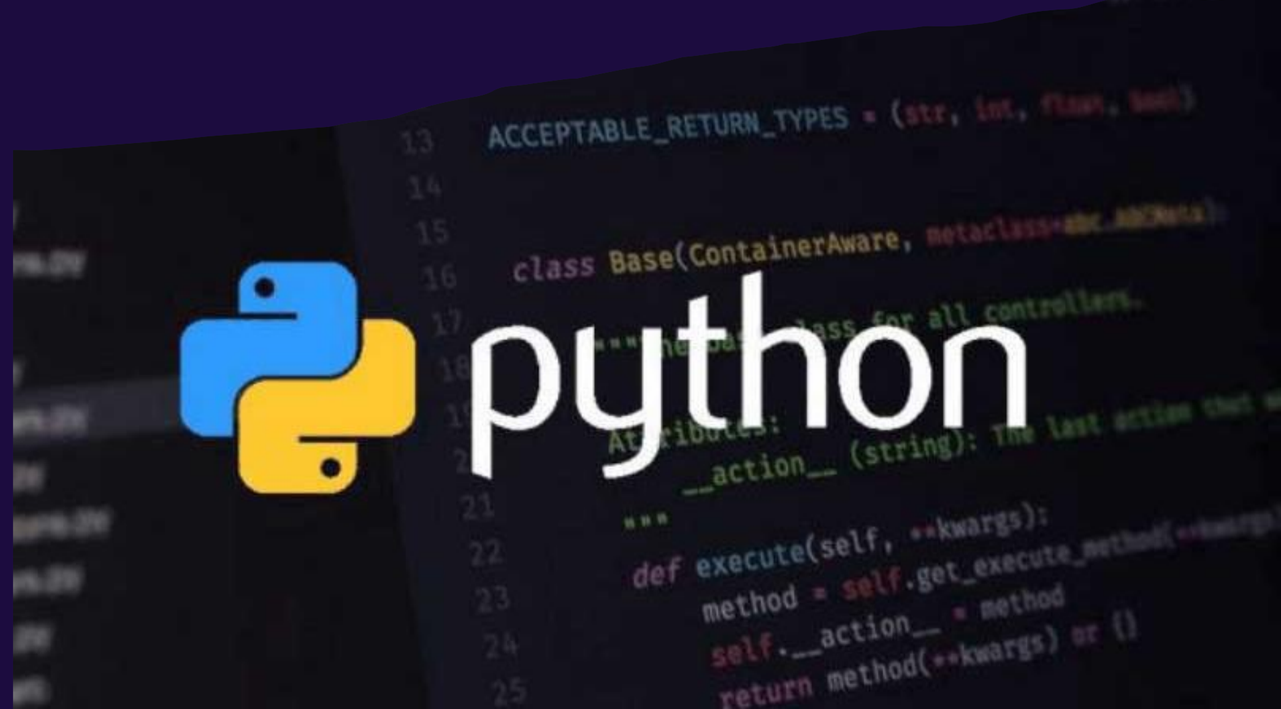
Hora da prática

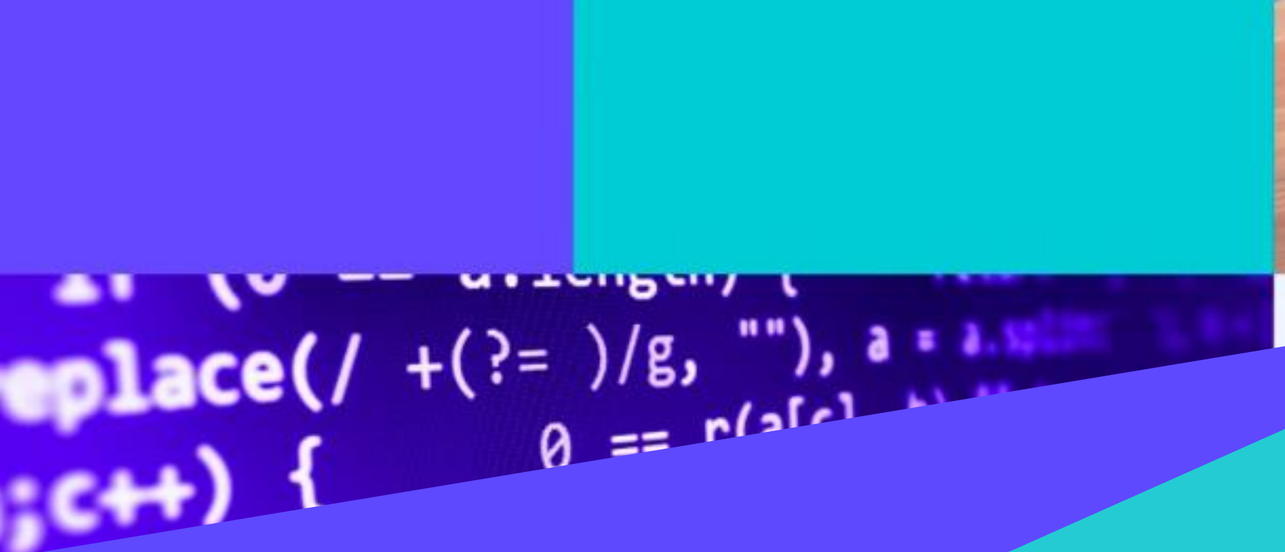
Desafio:

Escrever um programa que recebe os produtos a serem comprados e a forma de pagamentos escolhida, e de acordo com a forma de pagamento efetuar a compra utilizando o correto meio de pagamento.



Não se esqueça de commitar
Os programas desenvolvidos no seu Git =D





Obrigado pela participação

Esperamos que você tenha gostado, fique à vontade para nos enviar seus feedbacks sobre esta aula.
Esperamos você na próxima live, terça-feira 17/05 as 19:30. Não se esqueça de consultar o calendário e anote na sua agenda.

Se inscreva no canal e siga-nos nas rede sociais:

www.youtube.com/c/Devaria

@devaria_oficial @rafamazzucato @castellodaniel @oliveirandouglas