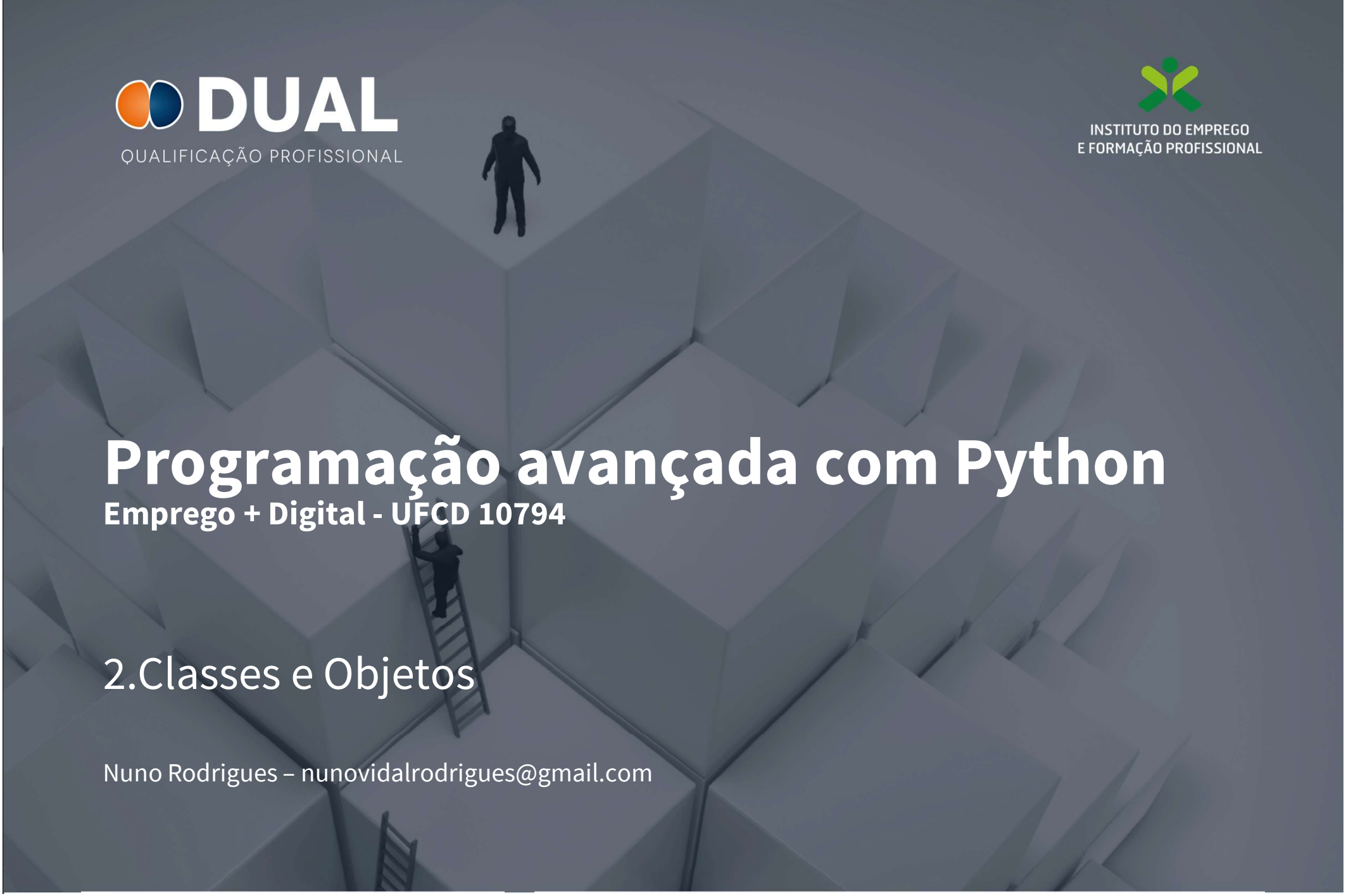


DUAL
QUALIFICAÇÃO PROFISSIONAL

INSTITUTO DO EMPREGO
E FORMAÇÃO PROFISSIONAL

Programação avançada com Python

Emprego + Digital - UFCD 10794

2.Classes e Objetos

Nuno Rodrigues – nunovidalrodrigues@gmail.com

UM SERVIÇO
AHK Deutsch-Portugiesische Industrie- und Handelskammer
Câmara de Comércio e Indústria Luso-Alemã

DGERT ENTIDADE FORMADORA CERTIFICADA

PRR Plano de Recuperação e Resiliência

REPÚBLICA PORTUGUESA

Financiado pela União Europeia
NextGenerationEU

40 ANOS 1983 – 2023
HISTÓRIAS DE QUALIFICAÇÃO

DUAL
QUALIFICAÇÃO PROFISSIONAL

Introdução às classes e objetos

Em programação é possível definir tipos de dados – classes – que permitem encapsular atributos e funcionalidades sendo possível a criação de novos tipos de variáveis. As classes são instanciáveis em objetos que são representados por variáveis do tipo das respetivas classes.

Os programas orientados por objetos são constituídos por conjuntos de classes. Os objetos interagem e solicitam uns aos outros elementos para executarem as respetivas funcionalidades. Na sua base, as classes escondem a implementação dos seus atributos e funcionalidades, no entanto permitem expor a informação necessária para que terceiros possam invocar a execução das suas próprias funcionalidades.

Classes e objetos

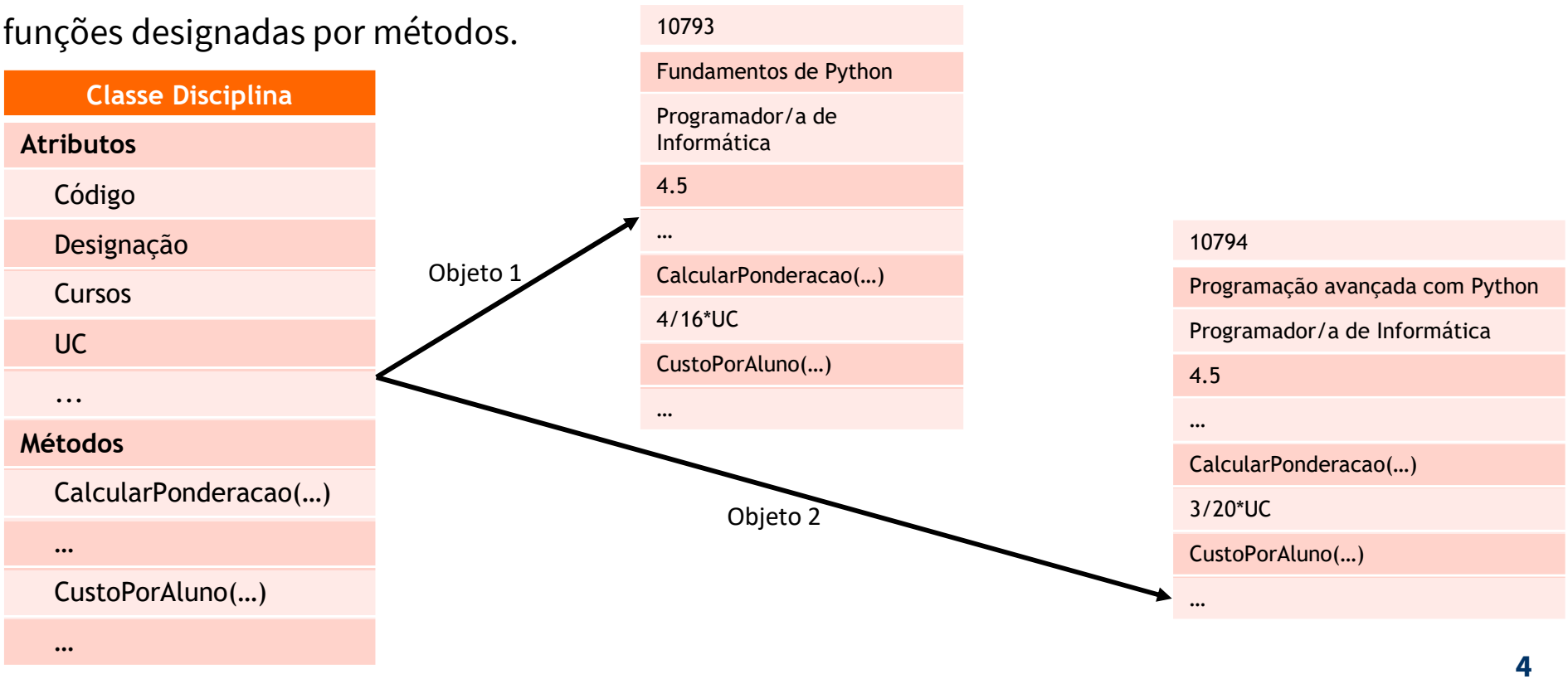
Entende-se por classe como sendo uma estrutura de dados que define, de forma abstrata, os atributos e as funcionalidades incorporadas numa família de objetos.

Supondo que pretendo criar uma classe “Disciplina” que representa de forma genérica todas as disciplinas académicas de um curso e possui atributos. Os atributos que possui são o seu “código”, a “designação”, o “curso a que pertence”, o “número de unidades de crédito”, a “sinopse” e o “conteúdo programático”, bem como funcionalidades que permitem determinar a ponderação da disciplina na classificação final do curso, taxa de crescimento de inscrições em relação a anos anteriores, o rácio do custo da disciplina por aluno inscrito, etc.

A disciplina “Programação” pode ser uma concretização da classe “Disciplina”, que contem todos os atributos e funcionalidades definidas naquela classe.

Classes e objetos

Os atributos ou propriedades são dados de diversos tipos, estruturas de dados ou tipos definidos pelo utilizador que caracterizam todos os objetos pertencentes a uma classe. As funcionalidades que possuem são as operações que esses objetos podem executar e são implementadas por funções designadas por métodos.



Variáveis de instante e estáticas

No que concerne aos objetos, os seus atributos são guardados em variáveis de instante. Estas variáveis são sempre atributos do objeto corrente e por esse motivo o seu nome é precedido de *self.* Contudo, as classes podem também conter variáveis estáticas, o que significa que fazem parte da classe embora não sejam atributos dos objetos.

Uma classe pode conter uma variável inteira que conte o número das suas instancias. As variáveis estáticas, como as de instante, podem ser de tipos primitivos, estruturas e dados ou tipos definidos pelo utilizador.

DUAL
QUALIFICAÇÃO PROFISSIONAL

Visibilidade das variáveis

Tanto as variáveis de instante como as variáveis estáticas são públicas, isto é, os seus valores podem ser lidos ou manipulados fora das classes que as encapsulam.

Esta manipulação de variáveis pode gerar erros, e para evitar esses erros, é possível esconder as variáveis não as tornando diretamente visíveis a outras classes ou módulos. Desta forma, as variáveis cujo nome é precedido de duplo *underscore* (_) não são visíveis por outras classes ou módulos. Existe ainda a possibilidade de ocultar menos as variáveis, utilizando pra isso um só *underscore.*

Embora seja possível ocultar estas variáveis, é sempre possível a sua manipulação por terceiros.

DUAL
QUALIFICAÇÃO PROFISSIONAL

Visibilidade das variáveis

Se observarmos o exemplo seguinte, temos a classe *EstudanteInf* define três variáveis de instante – *Nome*, *Teste1* e *Teste2* – encontram-se ocultas.

```
class EstudanteInf():
        def__init__(self,N,T1,T2):
                self.__Nome = N
                self.__Teste1 = T1
                self.__Teste2 = T2
```

Métodos de instante e métodos estáticos

Os métodos de instante definem funcionalidades dos objetos, portanto, só podem ser invocados se se associarem aos respetivos objetos. O primeiro parâmetro de um método de instante é sempre self, ou seja, o objeto corrente. No exemplo seguinte, demonstra-se um método de instante para calcular a classificação final de um aluno de informática que realizou dois testes.

```
def ClassFinal(self):
        return math.floor((self.Teste1+self.Teste2)/2+0,5)
```

As classes podem também conter métodos estáticos, isto é, métodos que fazem parte da classe, mas que não se referem aos objetos. Por exemplo, uma classe pode conter métodos para atribuir valores ou imprimir os valores das variáveis estáticas.

DUAL
QUALIFICAÇÃO PROFISSIONAL

Métodos de instante e métodos estáticos

O recurso a um método estático para atribuir uma disciplina o número total de alunos inscritos.

```
def TotalAlunos(N):
        EstudanteInf.TotalAlunos=N
```

Os métodos de instante têm de ser invocados para um objeto da classe que os define, enquanto os métodos estáticos são invocados para a classe que os contém. No exemplo seguinte, demonstra-se a invocação do método estático `TotalAlunos` da classe `EstudanteInf.`

```
print(f'{E.Nome} teve a classificação final de {E.ClassFinal()} valores')
        EstudanteInf.TotalAlunos(150)
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Construtores

A instanciação de objetos consiste na invocação automática de um método que constrói o objeto, colocando-o num estado inicial em memória principal. Por exemplo, veja-se a instanciação do estudante de informática João Silva que teve 14 e 16, respetivamente, no Teste 1 e no Teste 2:

```
E=EstudanteInf('João Silva', 14, 16)
```

Os construtores são métodos de inicialização. A assinatura dos construtores tem o seguinte formato:

```
def__init__(self, p1, p2, … , pn):
```

Onde p1, p2, … , pn é a lista de parâmetros.

DUAL
QUALIFICAÇÃO PROFISSIONAL

Construtores

O construtor da classe EstudanteInf tem três variáveis de instante, Nome, Teste1 e Teste2.

```
def __init__(self,N,T1,T2)
        self.Nome = N
        self.Teste1 = T1
        self.Teste2 = T2
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Acessos e propriedades

As variáveis de instante e as variáveis estáticas podem ser manipuladas por qualquer método de qualquer classe através dos seus acessos públicos. Estes são métodos para ler e alterar os valores das variáveis de instante e das variáveis estáticas cuja visibilidade foi escondida.

Regra geral, as classes definem, para cada uma das suas variáveis de instante, um par de acessos públicos:

- ✓ Um acesso para leitura – *getter* – que devolve o valor corrente da variável de instante e que pode também incluir código de formatação de saída do dado;
- ✓ Um acesso para atribuir um novo valor – *setter* - à variável de instante e que pode também incluir código de validação de dados.

DUAL
QUALIFICAÇÃO PROFISSIONAL

Acessos e propriedades

```
class EstudanteInf:
    def __init__ (self, N, T1, T2):
        self.__Nome = N
        self.__Teste1 = T1
        self.__Teste2 = T2
    def getNome(self):
        return self.__Nome
    def setNome(self,N):
        self.__Nome = N
```

Este par de acessos pode ser substituído por uma propriedade com as duas funcionalidades de setter e getter, como se exemplifica a seguir, recorrendo-se a uma propriedade para ler e alterar o valor da variável de instante Nome de EstudadanteInf.

DUAL
QUALIFICAÇÃO PROFISSIONAL

Acessos e propriedades

```
class EstudanteInf:
    def __init__(self,N,T1,T2):
        self.__Nome = N
        self.__Teste1 = T1
        self.__Teste2 = T2

    def getNome(self):
     return self.__Nome

    def setNome(self,N):
        self.__Nome=N
    @property
    def Nome(self):
        return self.__Nome

    @Nome.setter
    def Nome(self,N):
        self.__Nome=N
```

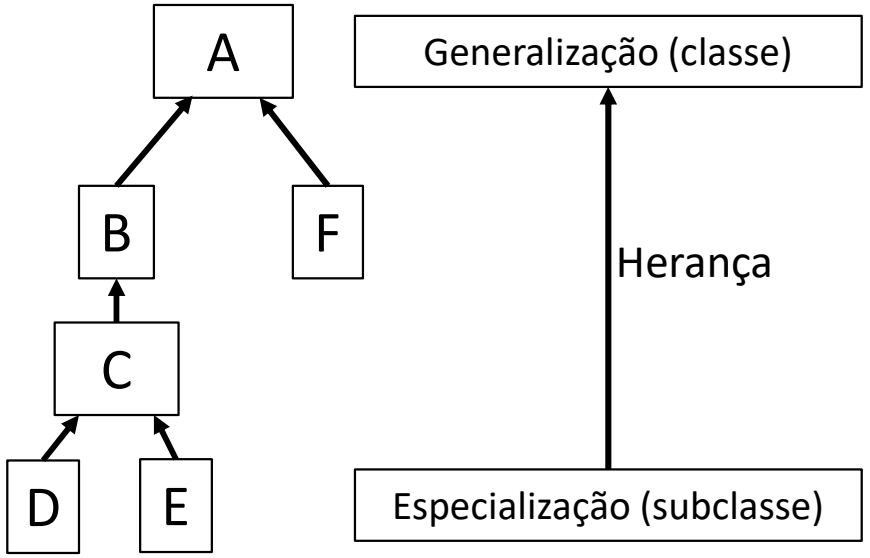
Assim, o invocador externo passa a tratar o par de acessos como se fossem um só atributo visível do objeto.

```
if __name__=="__main__":
    E=EstudanteInf("",0,0)
    E.Nome=input("Digite o nome correto do aluno")
    print (E.Nome)
```

Herança

As classes são estruturadas hierarquicamente em superclasses, classes e subclasses por recurso à herança. As classes herdam as propriedades e os métodos das superclasses de que derivam acrescentam mais propriedades e métodos e passam tudo às subclasses que delas derivarem. Esta hierarquia de classes é representada graficamente por árvores.

Uma vez que uma subclasse é uma especialização da superclasse de que deriva, as árvores de classes mostram os diferentes níveis de generalização.



Herança

No exemplo seguinte, demonstra-se um construtor de uma subclasse B que herda os atributos e métodos classe A.

```
class B(A):
    def __init__(self,a,p,n):
        A.__init__(self,a,p)
        self.NomeB=n
```

O Python permite a herança múltipla, isto é, uma subclasse pode derivar de mais do que uma superclasse.

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Temperaturas em Celsius e Fahrenheit

Pretende-se que defina a classe Temperatura com duas variáveis de instante, C e F, que representam determinada temperatura em graus Celsius e em graus Fahrenheit.

Elabore Também um programa de teste que leia a temperatura em graus Celsius e a converta em graus Fahrenheit. (Graus Fahrenheit=1,8*Grau Celsius+32)

Variável	Designação	Tipo	Significado
Temperatura	C	Real	Temperatura em graus Celsius.
Temperatura	F	Real	Temperatura em graus Fahrenheit.

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Temperaturas em Celsius e Fahrenheit

Construtor e acessos da classe Temperatura

```
class Temperatura():
    def __init__(self,C ):
        self.__C=C
        self.__F=1.8*C+32;
    def LerC(self):
            return f"{self.__C:>2.2f}"
    def LerF(self):
        return f"{self.__F:<2.2f}"
    def ImprimirTemp(self) :
        print(f"{self.LerC()}C={self.LerF()}F")
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Temperaturas em Celsius e Fahrenheit

Programa principal com instanciação de classe

```
class Temperatura():
    def __init__(self,C ):
        self.__C=C
        self.__F=1.8*C+32;
    def LerC(self):
            return f"{self.__C:>2.2f}"
    def LerF(self):
        return f"{self.__F:<2.2f}"
    def ImprimirTemp(self) :
        print(f"{self.LerC()}C={self.LerF()}F")

if __name__=='__main__':
    Temp=float(input("Digite a temperatura em graus Celsius "))
    T =Temperatura(Temp)
    T.ImprimirTemp()
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Aumento de preço unitário

Pretende-se um programa que permita a atualização de preços de produtos sempre que há uma subida percentual do preço unitário. Para a realização deste exercício será necessário recorrer a uma classe com a variável aumento e o método para lhe atribuir valores para que seja possível atualizar o valor dos produtos.

Variável	Designação	Tipo	Significado
Estática	Aumento	Inteira	Aumento percentual dos preços unitários

Inicialmente teremos apenas três produtos

```
Prod=(('LP1', 1, 10),('LP2', 2, 10),('LP3', 1.5, 15) )
Perc=5
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Aumento de preço unitário

Construtor e acessos da classe Produto

```
class Produto():
    Aumento=0
    def __init__(self, Des, Quant, Pu):
        self.__Des = Des #Designação
        self.__Quant = Quant #Quantidade
        self.__Pu=Pu #Preço Unitário
    @staticmethod
    def AumentoP(Perc):
        Produto.Aumento=Perc #Percentagem de aumento
        return
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

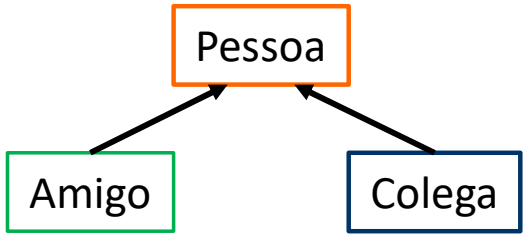
Exercício Aumento de preço unitário

Programa principal com a classe incluída.

```
class Produto():
    Aumento=0
    def __init__(self, Des, Quant, Pu):
        self.__Des = Des #Designação
        self.__Quant = Quant #Quantidade
        self.__Pu=Pu #Preço Unitário
    @staticmethod
    def AumentoP(Perc):
        Produto.Aumento=Perc #Percentagem de aumento
        return
    def ValorProduto(self):
       return self.__Quant*self.__Pu*(1+Produto.Aumento/100)
    @staticmethod
    def Cabecalho():
        print(f"{'Designação':^15}{'Valor (€)':^10}")
        return
    def Impressao(self):
        print(f"{self.__Des:<15} {self.ValorProduto():^8}")
        return
if  __name__=='__main__':
    Prod=(('LP1', 100, 10),('LP2', 200, 10),('LP3', 150, 15) )
    Perc=5
    Produto.AumentoP(Perc)
    Produto.Cabecalho()
    for P in Prod:
        Pr=Produto(P[0],P[1],P[2])
        Pr.Impressao()
```

Exercício Hierarquia de classes

Pretende-se que defina a classe Pessoa e as subclasses Amigo e Colega que derivam de Pessoa:



A **classe Pessoa** encapsula:

- ✓ Duas variáveis de instante, nome e telefone;
- ✓ Construtor;
- ✓ Acesso para ler os valores das variáveis de instante.

A **subclasse Amigo** encapsula:

- ✓ Duas variáveis de instante, local e ano de inicio da amizade;
- ✓ Construtor;
- ✓ Acesso para ler os valores das variáveis de instante.

A **subclasse Colega** encapsula:

- ✓ Duas variáveis de instante, local de trabalho e profissão;
- ✓ Construtor;
- ✓ Acesso para ler os valores das variáveis de instante.

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Hierarquia de classes

Classe Pessoa guardada em Class_Pessoa.py

```
class Pessoa:
    def __init__(self, N, T):
        self.__Nome = N
        self.__Telef = T
    def LerNome(self):
        return self.__Nome
    def LerTelef(self):
        return self.__Telef
    def Impressao(self):
        return f"{self.LerNome()} -- {self.LerTelef()} --"
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Hierarquia de classes

Subclasse Amigo guardada em Class_Amigo.py

```
from HPessoa import *
class Amigo(Pessoa):
    def __init__(self, N, T, L, A):
        super().__init__(N, T)
        self.__Local=L
        self.__Ano=A
    def LerLocal(self):
        return self.__Local
    def LerAno(self):
        return self.__Ano
    def ImpressaoA(self):
        print(super().Impressao(),self.LerLocal(),"--", self.LerAno())
        return
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Hierarquia de classes

Subclasse Colega guardada em Class_Colega.py

```
from HPessoa import *
class Colega(Pessoa):
    def __init__(self, N, T, P, LT):
        super().__init__(N, T)
        self.__Profissao=P
        self.__LocalTab=LT
    def LerProf(self):
        return self.__Profissao
    def LerLTrab(self):
        return self.__LocalTab
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Hierarquia de classes

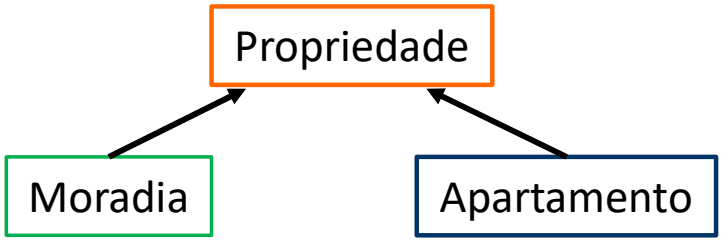
Programa principal com instanciação de classes

```
from HPessoa import *
from HAmigo import *
from HColega import *
if  __name__=='__main__':
    A=Amigo('Rita', 936666145 ,'Porto’,2022)
    C=Colega('Pedro',917675522 ,'Engenheiro', ‘ISEP')
    print(A.LerNome(), A.LerTelef(), A. LerLocal(), A.LerAno())
    print(C.LerNome(), C.LerTelef(), C. LerProf(), C.LerLTrab())
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Hierarquia de propriedades

Pretende-se que defina a classe Propriedade, e as subclasses Moradia e Apartamento que derivam de Propriedade:



E a sua implementação em Python nos ficheiros Propriedade.py, Moradia.py e Apartamento.Py.

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Hierarquia de Propriedades

Propriedade.py

```
class Propriedade():
    def __init__(self, NomeProp, NF):
        self.__Proprietario=NomeProp
        self.__NFiscal = NF
    @property
    def Proprietario(self):
        return self.__Proprietario
    @Proprietario.setter
    def Proprietario(N):
        self.__Proprietario=N
    @property
    def NFiscal(self):
        return self.__NFiscal
    @NFiscal.setter
    def NFiscal(NF):
        self. __NFiscal=NF
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Hierarquia de Propriedades

Moradia.py

```
from Propriedade import *
class Moradia(Propriedade):
    def __init__(self, P, NF, L, C):
        super().__init__(P, NF)
        self.__Local=L
        self.__Categ = C
    @property
    def Local(self):
        return self.__Local
    @Local.setter
    def Local(self, L):
        self. __Local=L
    @property
    def Categ(self):
         return self.__Categ
    @Categ.setter
    def Categ(self, C):
        self.__Categ=C
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Hierarquia de Propriedades

Apartamento.py

```
from Propriedade import *
class Apartamento(Propriedade):
    def __init__(self, P, NF, T, A):
        super().__init__(P, NF)
        self.__Tipo=T
        self.__Area = A
    @property
    def Tipo(self):
        return self.__Tipo
    @Tipo.setter
    def Tipo(self, T):
        self. __Tipo=T
    @property
    def Area(self):
         return self.__Area
    @Area.setter
    def Area(self, A):
        self.__Area=C
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

Exercício Hierarquia de Propriedades

Programa principal com instanciações de classes: Programa de teste que instancia uma moradia e um apartamento e imprime as respetivas variáveis de instante

```
from Propriedade import *
from Moradia import *
from Apartamento import *
if __name__=="__main__":
    M=Moradia("José Alves", 147098123, "Porto", "B")
    A=Apartamento("Adelina Costa", 123567890, "T3", 200)
    print(M.Proprietario, M.NFiscal, M. Local, M.Categ)
    print(A.Proprietario, A.NFiscal, A. Tipo, A.Area)
```

DUAL
QUALIFICAÇÃO PROFISSIONAL

INSTITUTO DO EMPREGO
E FORMAÇÃO PROFISSIONAL

Centro DUAL Lisboa
Avenida Infante D. Henrique,
Lote 320, Entreposto 2,
Piso 2, Fração 2
1800-220 Lisboa
Tel. +351 213 474 415
duallisboa@dual.pt

Centro DUAL Porto
Avenida Sidónio Pais, 379
4100-468 Porto
Tel. +351 226 061 561
dualporto@dual.pt

Centro DUAL Portimão
Rua Jaime Palhinha
Edifício Portimar, 2º Andar
8500-833 Portimão
Tel. +351 282 484 703
dualportimao@dual.pt

- **Qualificação Inicial**
- **Qualificação Contínua**
- **Qualificação Intraempresa**
- **Qualificação REFA**
- **Estágio nas Empresas**
- **Serviço às Empresas**
- **Centro de Exames TELC**

www.dual.pt

- DUAL.QualificacaoProfissional
- @dual_qp
- company/dual-ccila
- @dual_qp
- DUALQualificaçãoProfissional

40 ANOS 1983 - 2023
HISTÓRIAS DE QUALIFICAÇÃO

UM SERVIÇO
AHK Deutsch-Portugiesische Industrie- und Handelskammer
Câmara de Comércio e Indústria Luso-Alemã

ENTIDADE FORMADORA CERTIFICADA

PRR Plano de Recuperação e Resiliência

REPÚBLICA PORTUGUESA

Financiado pela União Europeia
NextGenerationEU