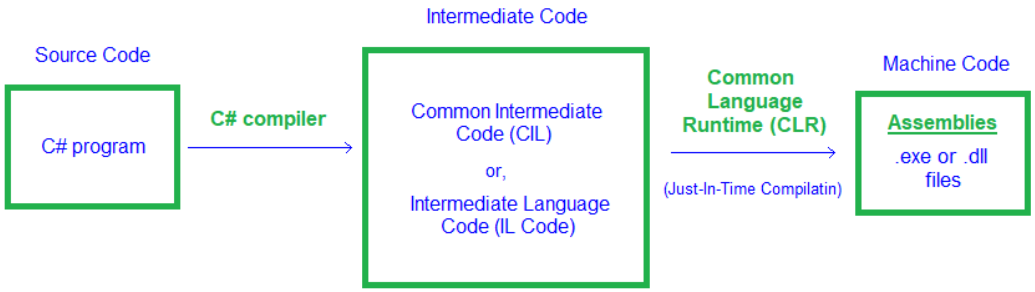
De Python para C#

Ao contrário dos programas em Python, que são interpretados no momento da sua execução, em C# os programas são compilados para um ficheiro executável. Assim para além do código fonte (ficheiro com extensão .cs) ficamos com um executável (ficheiro com extensão .exe) que é o programa que pode ser executado sem o código fonte. Para compilar um programa não podem existir erros no código fonte, ainda assim podem ocorrer erros durante a execução do programa.

Os programas em C# não são compilados diretamente para uma linguagem máquina, mas para uma linguagem intermédia designada por Common Intermediate Language (CIL). Esta linguagem é depois executada por uma camada de software conhecida por .Net Framework que funciona de forma independente do sistema operativo.



A última versão do .Net é multiplataforma permitindo que um programa possa ser executado em diferentes sistemas operativos e ambientes: Windows, MacOS, Linux, Android, Xbox, Web, etc.

Para criar programas em C# necessitamos de um editor de texto e um compilador, sendo mais fácil utilizar um IDE (Integrated Development Environment) que inclui as duas coisas como por exemplo o Visual Studio Community.

Também é possível utilizar o VS Code sendo necessário instalar o .Net de forma a poder criar, compilar e executar os programas através da linha de comandos.

A Microsoft define o .Net como sendo "uma palataforma de desenvolvimento gratuito, multiplataforma e de código aberto para criação de muitos tipos de aplicações."

O C# é uma linguagem de programação orientada aos objetos (OOP), apesar do Python também suportar esse princípio de programação não é obrigatório.

Definição de Variáveis

Python

x = 10
y = "Olá, Mundo!"

C#
Em C#, é necessário declarar o tipo de dados da variável. O C# é uma linguagem de programação fortemente tipada.

int x = 10;
string y = "Olá, Mundo!";

C# built-in **value** types (só os principais):

Tipo	**Exemplo**
bool	bool verdadeiro=true
byte	byte x=10
char	char letra='a'
decimal	decimal preco=10.5
double	double x=0.12
float	float x=0.13
int	int i=10
long	long z=10

C# built-in **reference** types (só os principais):

Tipo	**Exemplo**
object	object obj=new object()
string	string nome="Joaquim"
dynamic	dynamic x=10
class	Os objetos definidos pelo programador

As strings são um caso especial de um tipo de dados que é reference mas que o C# trata quase como se fosse um value type, assim os operadores == e != funcionam para testar os valores e não as referencias.

As strings são imutáveis isto significa que sempre que uma string é alterada na realidade é criada uma nova.

As linhas de código em C# terminam com ;

Operadores aritméticos e de atribuição

Operador	Descrição	Exemplo
+	Adição	x+y
-	Subtração	x-y
*	Multiplicação	x*y
/	Divisão	x/y
%	Resto da divisão inteira	x%y
++	Incrementar	x++
--	Decrementar	x--
=	Atribuição	x=10
+=	Soma e atribuição	x+=1
-=	Subtração e atribuição	x-=1
*=	Multiplicação e atribuição	X*=2
/=	Divisão e atribuição	x/=2

Operadores lógicos

Operador	Descrição	Exemplo
&&	E Lógico	X<5 && x>1
||	Ou Lógico	X==10 || X == 11
!	Negação	!(X==10 || X==11)

Operadores de comparação

Operador	Descrição	Exemplo
==	Igual a	X == Y
!=	Diferente de	X != Y
>	Maior do que	X > Y
<	Menor do que	X < Y
>=	Maior ou igual do que	X >= Y
<=	Menor ou igual do que	X <= Y

Estruturas Condicionais (If)

Python

```
if x > 5:
   print("x é maior que 5")
elif x == 5:
   print("x é igual a 5")
else:
   print("x é menor que 5")
```

C#

Em C#, os blocos de código são delimitados por chavetas `{}` e é necessário usar `Console.WriteLine` para escrever na consola. O alinhamento do código não condiciona a sua execução só facilita a sua leitura. As condições têm de estar entre parêntesis curvos ( )

```
if (x > 5)
{
   Console.WriteLine("x é maior que 5");
}
else if (x == 5)
{
   Console.WriteLine("x é igual a 5");
}
else
{
   Console.WriteLine("x é menor que 5");
}
```

Ciclos (Loops) - For

Python

```
for i in range(0, 5):
  print(i)
```

C#

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine(i);
}
```

O ciclo for em C# é composto por 3 partes: for(1ª parte; 2ª parte; 3ª parte)

A primeira parte é executada uma vez quando o ciclo é iniciado, a 3ª parte é executada sempre que termina uma execução (iteração) do ciclo, a 2ª parte é a condição que determina a conclusão do ciclo quando é false, esta é testada no inicio do ciclo e de cada vez que termina uma iteração, depois de executar a 3ª parte.

Ciclos (Loops) - While

Python

```
i = 0
while i < 5:
  print(i)
  i += 1
```

C#

```
int i = 0;
while (i < 5)
{
  Console.WriteLine(i);
  i++;
}
```

No C# as instruções **continue** e **break** funcionam da mesma forma que no Python.

Foreach

Em C# podemos utilizar um ciclo foreach para percorrer uma coleção.

```
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

foreach (string i in cars)

{

 Console.WriteLine(i);

}
```

Neste exemplo a coleção é um array de strings.

Funções

Python

```
def soma(a, b):
   return a + b
```

C#

Em C#, é necessário especificar o tipo de retorno da função e o tipo de cada parâmetro.

```
public int Soma(int a, int b)
{
   return a + b;
}
```

Estrutura de uma Aplicação Simples

Python

Uma aplicação simples em Python não requer uma estrutura específica para programas pequenos.

```
def main():
   print("Olá, Mundo!")

if __name__ == "__main__":
   main()
```

C#

Em C#, uma aplicação para a consola simples é estruturada da seguinte maneira. Note que `Main` é o ponto de entrada da aplicação.

A nova versão do .Net "esconde" esta estrutura.
Como o C# é uma linguagem de programação orientada aos objetos o nosso programa necessita de pelo menos uma classe (class Program).

```
using System;

namespace MinhaAplicacao
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Olá, Mundo!");
      }
   }
}
```

A instrução using permite importar código de outros namespaces. O código em C# está organizado em namespaces (que normalmente correspondem a DLL) e para utilizarmos as funções que estes implementam é necessário indicar através da instrução using.

Classes e objetos

Uma classe é uma estrutura que combina dados (propriedades) com funções (métodos) que manipulam essas propriedades.

As linguagens de programação orientadas aos objetos (POO) dão prioridade aos dados. A solução encontra-se definindo primeiro os dados e depois as funções que manipulam esses dados.

Nas linguagens POO os programas decompõem os problemas em módulos que representam objetos do mundo real, estes objetos são conhecidos como classes. As classes são comos os tipos de dados pois estas permitem definir variáveis do seu tipo, as variáveis definidas a partir de uma classe são os **objetos** ou instâncias/cópias.

Linguagem de Programação Estruturada	
Tipo de Dados	Variáveis
int	**int** x; x=10;

Linguagem POO	
Classes	Objetos

class Carro{
…
}
	Carro novo = **new** Carro();

O conceito é semelhante ao dos dicionários do Python, mas para além das chaves que contém os dados também inclui as funções que manipulam esses dados e que definem o comportamento do objeto.

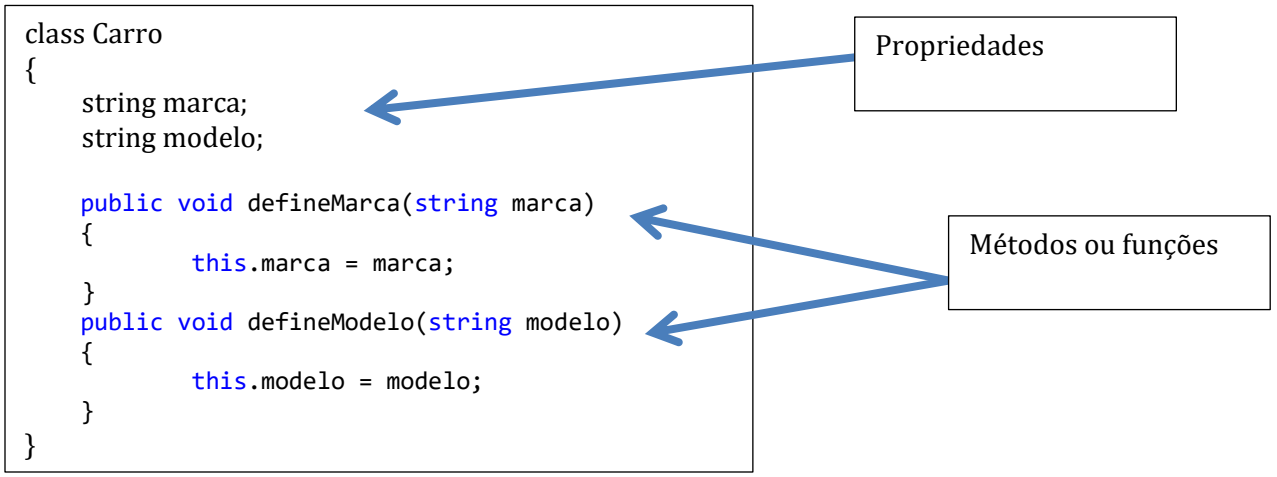
Conceito de Classe, Atributos, Métodos e eventos

Uma classe define um objeto com base nas suas propriedades (os campos ou variáveis) e as suas ações ou métodos que definem o seu comportamento (funções).

Assim um objeto tem dois tipos de membros, os dados e os métodos. Estes membros podem ter diferentes tipos de dados, incluindo outros objetos ou estruturas mais complexas como arrays, listas, dicionários ou outros.

A classe pode ainda herdar membros de outras classes, permitindo reutilizar classes criadas para servirem de base para outras classes.

Uma classe é um tipo de dados que depois permitirá definir variáveis (objetos) desse tipo.



Conceito de Objeto

Um objeto é uma variável criada a partir de uma classe. É o objeto que vai poder armazenar dados e executar funções sobre esses dados.

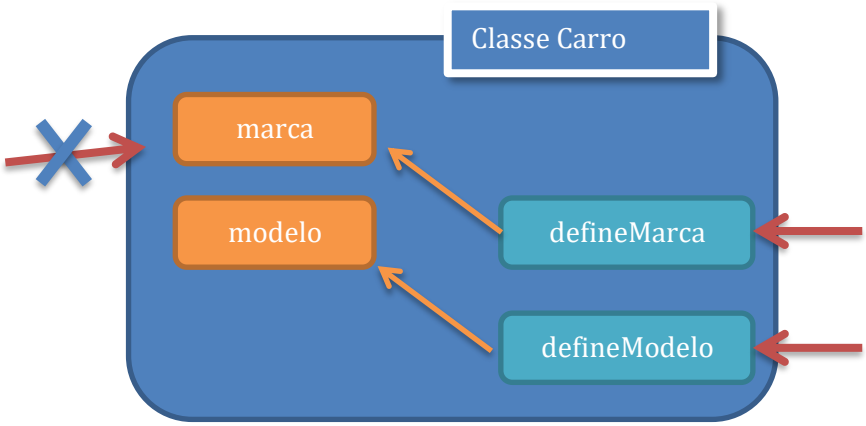
Ao criar um objeto a partir de uma classe estamos a criar uma cópia em memória das propriedades dessa classe, assim podem existir diferentes cópias da mesma classe bastando para isso definir diferentes objetos a partir da mesma classe. Estas cópias partilham o código, ou seja, as diferentes instâncias da classe só são diferentes pelos valores das suas propriedades, pois os métodos são os mesmos para a mesma classe.

```
Carro carro = new Carro();
Carro carro2 = new Carro();

carro.defineMarca("Ford");
carro2.defineMarca("BMW");
```

Conceito de Encapsulamento

A ideia do encapsulamento é que os dados de um objeto só devem ser manipulados por funções desse mesmo objeto. O objetivo é proteger os dados evitando erros validando os dados antes de os armazenar.



Conceito de visibilidade de classes, métodos e atributos

O conceito de encapsulamento está associado à visibilidade dos membros, assim os membros que definimos como privados (por defeito) só podem ser utilizados pelas funções dentro da classe. Os dados de uma classe devem ser privados, pois só devem ser alterados pelos métodos da classe.

As funções podem ser privadas ou publicas, pois, o objetivo é poderem ser chamadas por outras funções de outras classes.

As próprias classes podem ser definidas como visíveis por outros namespaces (public) ou só para uso interno do namespace do programa (internal).

No exemplo apresentado acima as propriedades são privadas (private), pois apesar de não ser indicado nenhum tipo de visibilidade por defeito estas são privadas, ao passo que as funções são publicas (foi indicado o nível de visibilidade public).

Em c# existem os seguintes níveis de visibilidade:

Nível de visibilidade/proteção	**Descrição**
private	Elemento só visível dentro da classe onde foi declarado
public	Elemento visível em qualquer classe
protected	Elemento visível dentro da classe onde foi declarado e nas classes derivadas
internal	Elemento visível em qualquer classe exceto fora do namespace
protected internal	Elemento visível dentro da classe onde foi declarado e nas classes derivadas, exceto fora do namespace

Construtores e destrutores

Um construtor é uma função com o mesmo nome da classe que é executada quando o objeto é criado. Esta função permite definir os valores iniciais do objeto.

O destrutor é uma função com o mesmo nome da classe mas com um ~ (til) no início do nome. Esta função é chamada automaticamente quando o objeto é destruído. O objetivo desta função é libertar recursos que o objeto alocou.

Na plataforma .Net existe um componente designado por Garbage Collector que se encarrega de alocar e libertar a memória sempre que os objetos já não são referenciados pelo programa.

Classes e membros estáticos

Uma classe estática é um conjunto de funções agrupadas dentro de uma classe. Neste caso não é possível criar objetos a partir desta classe, ou seja, não se pode instanciar a classe.

Todos os programas em C# têm uma classe estática com o nome Program que implementa a função Main, também estática, e é a partir desta classe e função que o código começa a ser executado.

Todos os outros elementos são tratados como classes, incluindo os formulários e os controlos dentro destes, porque na prática são classes.

Os membros estáticos de uma classe, mesmo numa classe que não é estática, são chamados pelo nome da classe e não pelo nome do objeto, isto porque na prática são membros da classe e independentes do objeto. No caso de propriedades estáticas estas são partilhadas pelos diferentes objetos.

Os membros estáticos podem ser utilizados para criar variáveis globais, desde que sejam públicas.

Nos módulos seguintes serão estudados conceitos mais avançados de POO como herança, polimorfismo e redefinição de métodos.