

The logo consists of the letters "GCONF" in a stylized, hand-drawn font. The letters are primarily black, but they are heavily textured with a vibrant, multi-colored spray paint effect. The colors transition through a rainbow spectrum, with visible streaks of red, orange, yellow, green, blue, and purple. The spray paint is most concentrated on the "G", "C", and "F", while the "O" and "N" have more solid black outlines.

INTRODUCTION

NEODYBLUE & TOOGY

The logo for GCONF is displayed again, but this time it includes a small, dark icon of a lit incandescent lightbulb positioned inside the letter "O". The rest of the logo, including the "G", "C", "N", and "F", is rendered in a bright, glowing pink color against a dark background.

Menu du jour

1 Programmation Impérative

Menu du jour

1 Programmation Impérative

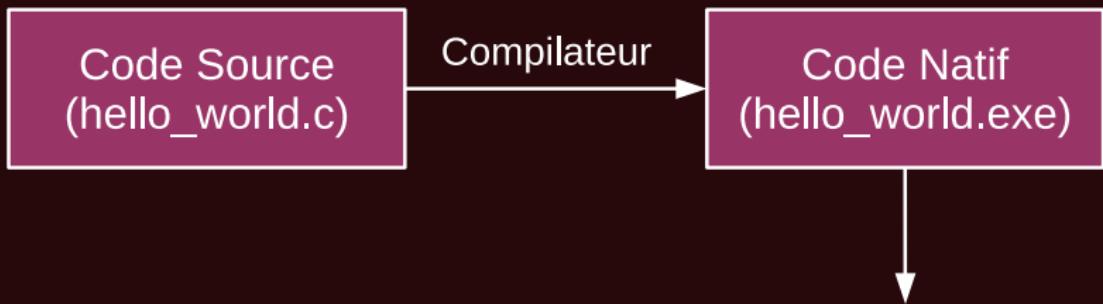
2 Syntaxe C#

Menu du jour

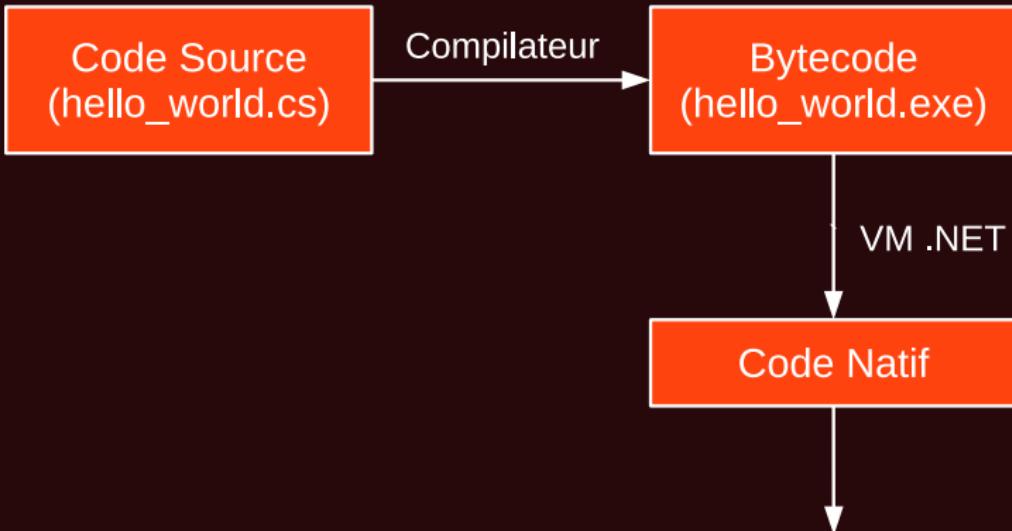
- 1** Programmation Impérative
- 2** Syntaxe C#
- 3** Programmation Orientée Objet (*POO*)

Menu du jour

- 1** Programmation Impérative
- 2** Syntaxe C#
- 3** Programmation Orientée Objet (*POO*)
- 4** Quelques conseils



Exécution par le processeur



Programmation **Impérative**

Mon ordinateur, mon esclave

- Fais moi un sandwich ;

Mon ordinateur, mon esclave

- Fais moi un sandwich ;
- Fais la vaisselle ;

Mon ordinateur, mon esclave

- Fais moi un sandwich ;
- Fais la vaisselle ;
- Sors les poubelles ;

Mon ordinateur, mon esclave

- Fais moi un sandwich ;
- Fais la vaisselle ;
- Sors les poubelles ;
- Tant que (le sol est sale) :
 { Lave le sol ; }

Mon ordinateur, mon esclave

- Fais moi un sandwich ;
- Fais la vaisselle ;
- Sors les poubelles ;
- Tant que (le sol est sale) :
 { Lave le sol ; }
- Si (il y a du courrier) :
 { Va chercher le courrier ; }

Mon ordinateur, mon esclave

- Fais moi un sandwich ;
- Fais la vaisselle ;
- Sors les poubelles ;
- Tant que (le sol est sale) :
 { Lave le sol ; }
- Si (il y a du courrier) :
 { Va chercher le courrier ; }

Il obéit.

Syntaxe C#

Les variables

Déclaration

- `<type> <name>;`
- `<type> <name> = <valeur>;`
- `<type>[] <name>;`
- `<type>[] <name> = new <type>[<size>];`

Déclaration

- `<type> <name>;`
- `<type> <name> = <valeur>;`
- `<type>[] <name>;`
- `<type>[] <name> = new <type>[<size>];`

Déclaration

- <type> <name>;
- <type> <name> = <valeur>;
- <type>[] <name>;
- <type>[] <name> = new <type>[<size>];

Déclaration

- <type> <name>;
- <type> <name> = <valeur>;
- <type>[] <name>;
- <type>[] <name> = new <type>[<size>];

Déclaration

Exemples

```
int i;  
string str;  
int a = 5;  
long verylong = 1000000000000000;  
float val = 6.4f;  
double d = 4.2;  
string str2 = "Hello";  
int[] datas = new int[10];
```

Manipulation

Nombres

```
int[] results = new int[2];
int a = 4;
bool vrai = true;
bool faux = false;
int b = 6;
a = -1;
results[0] = a + b; // [0] = 5
b = 3;
results[1] = a * b; // [1] = -3
```

Strings

```
string name = "Neodyblue";
string text = "Welcome " + name + " !"; // "Welcome Neodyblue!"
```

Manipulation

Nombres

```
int[] results = new int[2];
int a = 4;
bool vrai = true;
bool faux = false;
int b = 6;
a = -1;
results[0] = a + b; // [0] = 5
b = 3;
results[1] = a * b; // [1] = -3
```

Strings

```
string name = "Neodyblue";
string text = "Welcome " + name + " !"; // "Welcome Neodyblue!"
```

Les structures de contrôle

Les opérateurs de comparaison

- $3 == 3$
- $1 != 2$
- $1 < 3$
- $3 > 1$
- $1 <= 1$
- $4 >= 2$

Les opérateurs de comparaison

- $3 == 3$
- $1 != 2$
- $1 < 3$
- $3 > 1$
- $1 <= 1$
- $4 >= 2$

Les opérateurs de comparaison

- $3 == 3$
- $1 != 2$
- $1 < 3$
- $3 > 1$
- $1 <= 1$
- $4 >= 2$

Les opérateurs de comparaison

- $3 == 3$
- $1 != 2$
- $1 < 3$
- $3 > 1$
- $1 <= 1$
- $4 >= 2$

Les opérateurs de comparaison

- $3 == 3$
- $1 != 2$
- $1 < 3$
- $3 > 1$
- $1 <= 1$
- $4 >= 2$

Les opérateurs de comparaison

- $3 == 3$
- $1 != 2$
- $1 < 3$
- $3 > 1$
- $1 <= 1$
- $4 >= 2$

Le Test.

```
Console.WriteLine("Say my name!");  
string name = Console.ReadLine();  
  
if (name == "Heisenberg")  
{  
    Console.WriteLine("You're goddamn right.");  
}  
else  
{  
    Console.WriteLine("You lose!");  
}
```

Les boucles

```
for (<init> ; <boolean> ; <step>)
{
    // Do something
}
```

```
for (int i = 0; i < 10; ++i)
{
    // This code will be executed 10 times
    // i values : 0, 1, 2, ..., 8, 9
}
```

```
while (<boolean>)
{
    // Do something
}
```

```
string str = "";  
while (str.Length != 10)  
{  
    str += "A";  
}
```

```
do
{
    // Do something at least 1 time
} while (<boolean>);
```

```
string str = "";  
do  
{  
    str = Console.ReadLine();  
    // Do something with str  
} while (str != "exit");
```

Factorisation : les fonctions

Déclaration de fonction

```
<type> <name>(<parameter>)
{
    // Code
}
```

```
int sum(int a, int b)
{
    return a + b ;
}
bool is_even(int n)
{
    return n % 2 == 0 ;
}
```

```
int div(int c, int d)
{
    if (d == 0)
    {
        // FAIL
    }
    else
        return c / d ;
}
```

```
int sum(int a, int b)
{
    return a + b ;
}
bool is_even(int n)
{
    return n % 2 == 0 ;
}
```

```
int div(int c, int d)
{
    if (d == 0)
    {
        // FAIL
    }
    else
        return c / d ;
}
```

```
int sum(int a, int b)
{
    return a + b ;
}
bool is_even(int n)
{
    return n % 2 == 0 ;
}
```

```
int div(int c, int d)
{
    if (d == 0)
    {
        // FAIL
    }
    else
        return c / d ;
}
```

```
int x = 5;  
int y = 11;  
  
int result_sum = sum(x, y); // result_sum = 16  
int result_div = div(x, y); // result_div = 0  
bool even = is_even(y); // even = false
```

Surcharge

```
void print(int n)
{
    Console.WriteLine (n);
}

void print(int n, string msg)
{
    Console.WriteLine (msg + " " + n);
}
```

Les collections

Listes

```
List<<type>> <name> = new List<<type>>();
```

```
List<string> words = new List<string>();
```

Ajout et suppression

```
words.Add("hello");
```

```
words.Add("world");
```

```
words.Add("test");
```

```
words.Remove("test");
```

```
List<string> words = new List<string>();
```

Ajout et suppression

```
words.Add("hello");
```

```
words.Add("world");
```

```
words.Add("test");
```

```
words.Remove("test");
```

Parcourir une collection

```
foreach (<type> <name> in <collection>)
{
    // Do something with <name>
}

for (int i = 0; i < <collection>.Count ; ++i)
{
    // Do something with <name> and its index
}
```

Parcourir une collection

```
foreach (string word in words)
{
    Console.WriteLine(word);
}

for (int i = 0; i < words.Count; ++i)
{
    Console.WriteLine("[ " + i + " ]: " + words[i]);
}
```

Programmation Orientée Objet (*POO*)

Field (champ)

```
public class Circle
{
    public double radius;
    public string color;
}
```

Method

```
public class Circle
{
    public double radius;
    public string color;

    public void setColor(string newColor)
    {
        this.color = newColor;
    }
}
```

Property (propriété)

Snippet VS : propfull

```
public class Truc
{
    // backing field
    private int _attribut;

    public int Attribut // propriété
    {
        get { return _attribut; }
        set { _attribut = value; }
    }
}
```

Field (champ)

```
public class Circle
{
    public double radius;
    public string color;
}
```

Method

```
public class Circle
{
    public double radius;
    public string color;

    public void setColor(string newColor)
    {
        this.color = newColor;
    }
}
```

Property (propriété)

Snippet VS : propfull

```
public class Truc
{
    // backing field
    private int _attribut;

    public int Attribut // propriété
    {
        get { return _attribut; }
        set { _attribut = value; }
    }
}
```

Field (champ)

```
public class Circle
{
    public double radius;
    public string color;
}
```

Method

```
public class Circle
{
    public double radius;
    public string color;

    public void setColor(string newColor)
    {
        this.color = newColor;
    }
}
```

Property (propriété)

Snippet VS : propfull

```
public class Truc
{
    // backing field
    private int _attribut;

    public int Attribut // propriété
    {
        get { return _attribut; }
        set { _attribut = value; }
    }
}
```

Validation

```
class Thermostat
{
    private int _temperature; // backing field

    public int Temperature // propriété
    {
        get { return _temperature; }
        set
        {
            if (value >= 50)
                _temperature = 50;
            else
                _temperature = value;
        }
    }
}
```

Auto-propriété

Snippet VS : prop

```
public class Objet
{
    public int Attribut { get; set; }
}
```

Accès privé sur le set

Snippet VS : propg

```
public class Objet
{
    public int Attribut { get; private set; }
}
```

Auto-propriété

Snippet VS : prop

```
public class Objet
{
    public int Attribut { get; set; }
}
```

Accès privé sur le set

Snippet VS : propg

```
public class Objet
{
    public int Attribut { get; private set; }
}
```

```
public class Girlfriend
{
    private int _boobs_size;
    private string _name;

    public string Name {
        get { return _name; }
    }
}

public class Mira : Girlfriend
{
    private void ShowBoobs() {
        // Show something interesting
    }
}
```

```
public class Girlfriend
{
    private int _boobs_size;
    private string _name;

    public string Name {
        get { return _name; }
    }
}

public class Mira : Girlfriend
{
    private void ShowBoobs() {
        // Show something interesting
    }
}
```

Interface

```
interface IBicycle
{
    string BrandName { get; set; }

    void Brake();
}

class BlueBicycle : IBicycle
{
    private string _brandName;

    public string BrandName {
        get { return _brandName; }
        set { _brandName = lowerCase(value); }
    }

    void Brake() {
        // Braaaaaake! :o
    }
}
```

Interface

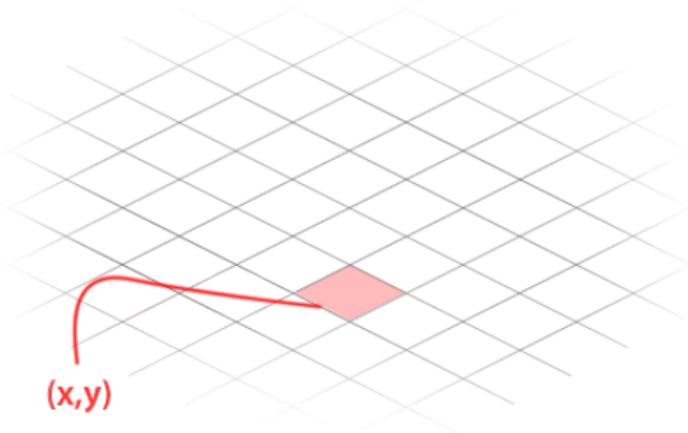
```
interface IBicycle
{
    string BrandName { get; set; }

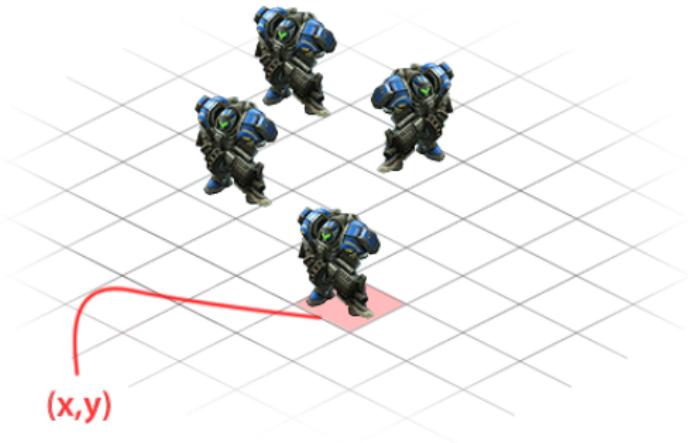
    void Brake();
}

class BlueBicycle : IBicycle
{
    private string _brandName;

    public string BrandName {
        get { return _brandName; }
        set { _brandName = lowerCase(value); }
    }

    void Brake() {
        // Braaaaaake! :o
    }
}
```





Modélisation d'une unité

```
class Unit
{
    public Tuple<int,int> Position { get; set; }

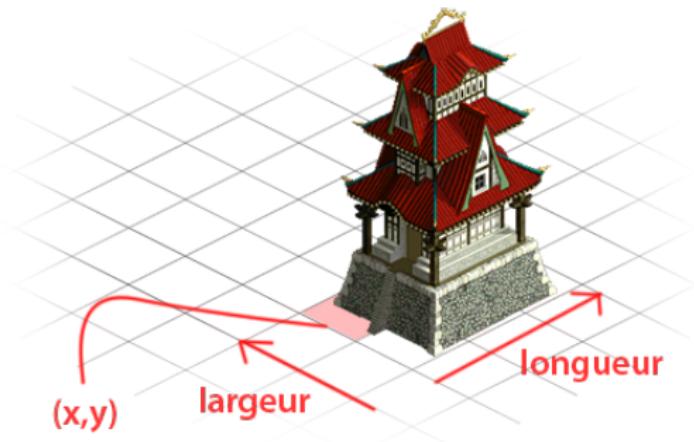
    public int HealthPoints { get; set; }
    private int _maxHealthPoints;

    public int Speed { get; private set; }

    public int Dps { get; private set; }

    public void Move(Tuple<int,int> destination)
    {
        // Bouger
    }

    public void Die()
    {
        // Mourir
    }
}
```



Modélisation d'un bâtiment

```
class Building
{
    public Tuple<int,int> Position { get; private set; }
    public Tuple<int,int> Size { get; private set; }

    public List<Unit> Units { get; set; }

    public int HealthPoints { get; set; }
    private int _maxHealthPoints;

    public void Die()
    {
        // Mourir
    }
}
```

```
class Unit
{
    public Tuple<int,int> Position { get; set; }

    public int HealthPoints { get; set; }
    private int _maxHealthPoints;

    public int Speed { get; private set; }

    public int Dps { get; private set; }

    public void Move(Tuple<int,int> destination)
    {
        // Bouger
    }

    public void Die()
    {
        // Mourir
    }
}

class Building
{
    public Tuple<int,int> Position { get; private set; }
    public Tuple<int,int> Size { get; private set; }

    public List<Unit> Units { get; set; }

    public int HealthPoints { get; set; }
    private int _maxHealthPoints;

    public void Die()
    {
        // Mourir
    }
}
```

```

abstract class GameItem
{
    public Tuple<int,int> Position
        { get; private set; }

    public int HealthPoints
        { get; set; }

    private int _maxHealthPoints;

    public abstract void Die();
}

class Building : GameItem
{
    public Tuple<int,int> Size
        { get; private set; }

    public List<Unit> Units
        { get; set; }

    public override void Die()
    {
        // Mourir
    }
}

```

```

class Unit : GameItem
{
    public int Speed
        { get; private set; }

    public int Dps
        { get; private set; }

    public void Move(
        Tuple<int,int> destination)
    {
        // Bouger
    }

    public override void Die()
    {
        // Mourir
    }
}

```

Héritage

```

abstract class GameItem
{
    public Tuple<int,int> Position
        { get; private set; }

    public int HealthPoints
        { get; set; }

    private int _maxHealthPoints;

    public abstract void Die();
}

class Building : GameItem
{
    public Tuple<int,int> Size
        { get; private set; }

    public List<Unit> Units
        { get; set; }

    public override void Die()
    {
        // Mourir
    }
}

```

```

class Unit : GameItem
{
    public int Speed
        { get; private set; }

    public int Dps
        { get; private set; }

    public void Move(
        Tuple<int,int> destination)
    {
        // Bouger
    }

    public override void Die()
    {
        // Mourir
    }
}

```

Héritage

```

abstract class GameItem
{
    public Tuple<int,int> Position
        { get; private set; }

    public int HealthPoints
        { get; set; }

    private int _maxHealthPoints;

    public abstract void Die();
}

class Building : GameItem
{
    public Tuple<int,int> Size
        { get; private set; }

    public List<Unit> Units
        { get; set; }

    public override void Die()
    {
        // Mourir
    }
}

```

```

class Unit : GameItem
{
    public int Speed
        { get; private set; }

    public int Dps
        { get; private set; }

    public void Move(
        Tuple<int,int> destination)
    {
        // Bouger
    }

    public override void Die()
    {
        // Mourir
    }
}

```

Héritage

```

abstract class GameItem
{
    public Tuple<int,int> Position
        { get; private set; }

    public int HealthPoints
        { get; set; }

    private int _maxHealthPoints;

    public abstract void Die();
}

class Building : GameItem
{
    public Tuple<int,int> Size
        { get; private set; }

    public List<Unit> Units
        { get; set; }

    public override void Die()
    {
        // Mourir
    }
}

```

```

class Unit : GameItem
{
    public int Speed
        { get; private set; }

    public int Dps
        { get; private set; }

    public void Move(
        Tuple<int,int> destination)
    {
        // Bouger
    }

    public override void Die()
    {
        // Mourir
    }
}

```

Héritage

```
class MyClass
{
    public void MyFonction()
    {
        List<GameItem> gameItems = new List<GameItem>();

        gameItems.Add(new Unit());
        gameItems.Add(new Building());
    }
}
```

Virtual methods

```
abstract class Unit
{
    public virtual void Die()
    {
        // A single death is a tragedy; a million deaths is a statistic.
    }
}

class Bomber : Unit
{
    public override void Die()
    {
        // Kill everyone around before actually dying

        base.Fonction(parameter);
    }
}
```

Virtual methods

```
abstract class Unit
{
    public virtual void Die()
    {
        // A single death is a tragedy; a million deaths is a statistic.
    }
}

class Bomber : Unit
{
    public override void Die()
    {
        // Kill everyone around before actually dying

        base.Fonction(parameter);
    }
}
```

Quelques conseils



<https://www.jetbrains.com/resharper/>

Documenter son code

```
/// <summary>
/// Elle fait quoi cette fonction?
/// </summary>
/// <returns>Qu'est ce que ça retourne ?</returns>
/// <param name="a">C'est quoi ce paramètre ?</param>
/// <param name="b">C'est quoi ce paramètre ?</param>
int sum(int a, int b)
{
    return a + b;
}
```



Des choses pas claires ?