Fonctions receivers

Exercice 1

```go
package main

import (
    "fmt"
    "time"
)

// Structure Car
type Car struct {
    brand string
    year  int
}

// Fonction receiver Age pour la structure Car
func (c Car) Age() int {
    return time.Now().Year() - c.year
}

func main() {
    myCar := Car{brand: "Toyota", year: 2010}
    fmt.Printf("La voiture a %d ans.", myCar.Age())
}
```

Exercice 2

```go
package main

import (
    "fmt"
    "time"
)

// Structure Car
type Car struct {
    brand string
    year  int
}

// Fonction receiver Age pour la structure Car
func (c Car) Age() int {
    return time.Now().Year() - c.year
}

func (c *Car) UpdateYear(newYear int) {
    c.year = newYear
}

func main() {
    myCar := Car{brand: "Toyota", year: 2010}
```

```
    fmt.Printf("La voiture a %d ans.", myCar.Age())
    myCar.UpdateYear(2009)
    fmt.Printf("La voiture a %d ans.", myCar.Age())
}
```

Exercice 3

```
package main

import (
    "fmt"
    "time"
)

// Structure Car
type Car struct {
    brand string
    year  int
}

// Fonction receiver Age pour la structure Car
func (c *Car) Age() int {
    return time.Now().Year() - c.year
}

func (c *Car) UpdateYear(newYear int) {
    c.year = newYear
}

func main() {
    myCar := Car{brand: "Toyota", year: 2010}
    fmt.Printf("La voiture a %d ans.", myCar.Age())
    myCar.UpdateYear(2009)
    fmt.Printf("La voiture a %d ans.", myCar.Age())
}

// L'utilisation des pointeurs permet de modifier l'instance originale de la structure,
// au lieu de travailler sur une copie. Cela est essentiel pour les modifications qui doivent
// persister en dehors de la portée de la méthode.
```

Exercice 4

```
package main

import (
    "fmt"
    "time"
)

// Structure Car
type Car struct {
    brand string
    year  int
```

```
}

// Fonction receiver Age pour la structure Car
func (c Car) Age() int {
    return time.Now().Year() - c.year
}

func (c *Car) UpdateYear(newYear int) {
    c.year = newYear
}

// Fonction receiver Equals pour la structure Car
func (c Car) Equals(other Car) bool {
    return c.brand == other.brand && c.year == other.year
}

func main() {
    myCar := Car{brand: "Toyota", year: 2010}
    fmt.Printf("La voiture a %d ans.", myCar.Age())
    myCar.UpdateYear(2009)
    fmt.Printf("La voiture a %d ans.", myCar.Age())

    // Comparaison de deux voitures
    myCar2 := Car{brand: "Toyota", year: 2010}
    fmt.Println(myCar.Equals(myCar2))
    myCar2.UpdateYear(2009)
    fmt.Println(myCar.Equals(myCar2))
}
```

Exercice 5

```
package main

import (
    "fmt"
    "time"
)

// Structure Car
type Car struct {
    brand string
    year  int
    color string
    engine int
}

// Fonction receiver Equals pour la structure Car
func (c Car) Equals(other Car) bool {
    return c.brand == other.brand && c.year == other.year
}

// Fonction receiver Age pour la structure Car
func (c Car) Age() int {
    return time.Now().Year() - c.year
}
```

```go
// Fonction receiver UpdateYear pour la structure Car
func (c *Car) UpdateYear(newYear int) *Car {
    c.year = newYear
    return c
}

// Fonction receiver UpdateColor pour la structure Car
func (c *Car) SetColor(newColor string) *Car {
    c.color = newColor
    return c
}

// Fonction receiver UpgradeEngine pour la structure Car
func (c *Car) UpgradeEngine() *Car {
    c.engine++
    return c
}


func main() {
    myCar := Car{brand: "Toyota", year: 2010}
    fmt.Printf("La voiture a %d ans.", myCar.Age())
    myCar.UpdateYear(2009)
    fmt.Printf("La voiture a %d ans.", myCar.Age())

    // Comparaison de deux voitures
    myCar2 := Car{brand: "Toyota", year: 2010}
    fmt.Println(myCar.Equals(myCar2))
    myCar2.UpdateYear(2009)
    fmt.Println(myCar.Equals(myCar2))

    // Mise à jour chaînée
    myCar.UpdateYear(2021).SetColor("red").UpgradeEngine()
    fmt.Println(myCar)
}
```