Tableaux et Tranches

Exercice 1

```go
package main

import "fmt"

func main() {
    names := []string{"Einstein", "Tesla", "Shepard"}
    distances := []int{50, 40, 75, 30, 125}
    data := []byte{'H', 'E', 'L', 'L', 'O'}
    ratios := []float64{3.14145}
    alives := []bool{true, false, true, false}
    zero := []byte{}

    fmt.Printf("names    : %[1]T %[1]v\n", names)
    fmt.Printf("distances: %[1]T %[1]d\n", distances)
    fmt.Printf("data     : %[1]T %[1]d\n", data)
    fmt.Printf("ratios   : %[1]T %.2[1]f\n", ratios)
    fmt.Printf("alives   : %[1]T %[1]t\n", alives)
    fmt.Printf("zero     : %[1]T %[1]d\n", zero)
}
```

Exercice 2

```go
package main

import (
"fmt"
"sort"
"strings"
)

func main() {
noms := []string{"Einstein", "Shepard", "Tesla"}

livres := []string{"Restez en Or", "Feu", "La Revanche de Kafka"}
sort.Strings(livres)

nums := [...]int{5, 1, 7, 3, 8, 2, 6, 9}
sort.Ints(nums[:])

fmt.Printf("%q\n", strings.Join(noms, " et "))
fmt.Printf("%q\n", livres)
fmt.Printf("%d\n", nums)
}
```

Exercice 3

```go
package main

import (
    "fmt"
    "sort"
    "strings"
)

func main() {
    namesA := "Da Vinci, Wozniak, Carmack"
    namesB := []string{"Wozniak", "Da Vinci", "Carmack"}

    namesC := strings.Split(namesA, ", ")

    sort.Strings(namesC)
    sort.Strings(namesB)

    if len(namesC) == len(namesB) {
        for i := range namesC {
            if namesC[i] != namesB[i] {
                return
            }
        }
        fmt.Println("Ils sont égaux.")
    }
}
```

Exercice 4

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    // ------------------------------------------------------------
    // Crée des slices vides
    // ------------------------------------------------------------

    // Pizza toppings
    var pizza []string

    // Departure times
    var departures []time.Time

    // Student graduation years
    var grads []int

    // On/off states of lights in a room
    var lights []bool

    // ------------------------------------------------------------
```

```
    // Ajoute des éléments aux slices
    // ------------------------------------------------------------
    pizza = append(pizza, "pepperoni", "onions", "extra cheese")

    now := time.Now()
    departures = append(departures,
        now,
        now.Add(time.Hour*24), // 24 hours after `now`
        now.Add(time.Hour*48)) // 48 hours after `now`

    grads = append(grads, 1998, 2005, 2018)

    lights = append(lights, true, false, true)

    // ------------------------------------------------------------
    // Affiche les slices
    // ------------------------------------------------------------

    fmt.Printf("pizza       : %s\n", pizza)
    fmt.Printf("\ngraduations : %d\n", grads)
    fmt.Printf("\ndepartures  : %s\n", departures)
    fmt.Printf("\nlights      : %t\n", lights)
}
```

Exercice 5

```
package main

import (
    "fmt"
    "strconv"
    "strings"
)

func main() {
    data := "2 4 6 1 3 5"
    splitted := strings.Fields(data)

    var nums []int
    for _, s := range splitted {
        n, _ := strconv.Atoi(s)
        nums = append(nums, n)
    }

    fmt.Println("nums        :", nums)

    evens, odds := nums[:3], nums[3:]

    fmt.Println("evens       :", evens)
    fmt.Println("odds        :", odds)

    fmt.Println("middle      :", nums[2:4])
    fmt.Println("first 2     :", nums[:2])
    fmt.Println("last 2      :", nums[len(nums)-2:])
```

```
    fmt.Println("evens last 1:", evens[len(evens)-1:])
    fmt.Println("odds last 2 :", odds[len(odds)-2:])
}
```

Exercice 6

```
package main

import "fmt"

func main() {func main() {
    var (
        nums   []int
        oldCap float64
    )

    // boucle 10 millions de fois
    for len(nums) < 1e7 {
        // obtient la capacité
        c := float64(cap(nums))

        // imprime seulement lorsque la capacité change
        if c == 0 || c != oldCap {
            // imprime également le ratio de croissance : c/oldCap
            fmt.Printf("len:%-15d cap:%-15g growth:%-15.2f\n",
                len(nums), c, c/oldCap)
        }

        // garde une trace de la capacité précédente
        oldCap = c

        // ajoute un élément arbitraire à la tranche
        nums = append(nums, 1)
    }
}
```