Contrôle d'entrainement de go

Ce sujet est donné à but d'entrainement, le contrôle comportera plus de questions et d'exercices.

Pour toute question ou remarque relative aux cours : pierre.bertin-johannet@orange.fr

Exercice 0: QCM

Plusieurs réponses possibles, pas de points négatifs.

Le langage go est ?

a) Interprété.
b) Compilé.

Soit la fonction déclarée comme suit:

```
func f(a int) float32
```

a) Le type de retour de f est int
b) Le type de retour de f est float32
c) La fonction f accepte un argument de type int
d) La fonction f accepte un argument de type float

Soit la fonction déclarée comme suit:

```
func f(a float32) (float32, string)
```

a) Le type de retour de f est int
b) Le type de retour de f est float32
c) La fonction f accepte deux arguments de type int et string
d) La fonction f renvoie deux valeurs de type int et float

Lorsqu'on utilise plus un objet en go

a) Il faut libérer la mémoire manuellement.
b) La mémoire sera libérée à la fin de la fonction.
c) La mémoire sera libérée quand l'objet ne sera plus accessible.

En go, une structure

a) Contient des attributs de types différents.
b) Ne peut pas être passée par pointeur à une fonction.
c) Peut être passée par copie à une fonction.

Une méthode en go

a) Doit avoir au moins deux arguments.
b) Ne peut être appelée que sur un objet.
c) Est définie par une syntaxe différente d'une fonction.

Une Interface

a) Contient des attributs et des méthodes.
b) Ne peut pas être utilisée dans les listes
c) Sert à implémenter les `map`
d) Est définie par une syntaxe différente d'une structure.

Une slice

a) Ne peut pas changer de taille.
b) Peut être passée par pointeur à une fonction.
c) Peut être passée par copie à une fonction.

Exercice 1

a) Qu'affichera le code suivant dans la console :

```
func main() {
    a := 6.0
    b := 2.5
    for i := 0; i < 3; i += 2 {
        b += a
        fmt.Printf("%f\n", b)
    }
}
```

a) Qu'affichera le code suivant dans la console :

```
func main() {
    values := []int{30, 20, 10, 40, 5}
    var a = 0
    var r = 0
    for i, v := range values {
        if a < v {
           r = i
           a = v
        }
    }
    fmt.Printf("%d\n", r)
}
```

a) Completez le code suivant afin que le programme affiche la somme des nombres dans le tableau de taille n.

```
func main() {
    values := [5]float64{/* flottants ici*/}
    // completez ici
```

Exercice 2

Complétez les types manquants dans ces fonctions.

```
func replaceEvens(arr ....., replacement .....) {
  for i := 0; i < len(arr); i++ {
    if arr[i]%2 == 0 {
      arr[i] = replacement
    }
  }
}
```

```
func split(chaine ....., delim .....) ..... {
  var result []string
  start := 0
  for i := 0; i < len(s); i++ {
    if string(s[i]) == delim {
      result = append(result, s[start:i])
      start = i + 1
    }
  }
  result = append(result, s[start:]) // Add the last segment
  return result
}
```

Exercice 3

Que va afficher le code suivant dans la console ?

```
func a(n int) {
    n = n + 2
}
func b(n *int, b int) {
    *n = b
}
func c(n *int) {
    *n = *n + 2
}
```

```go
func main() {
    i := 1
    j := 2
    k := 3

    a(k)
    b(&j, i)
    c(&k)

    fmt.Printf("%d, %d, %d\n", i, j, k)
}
```

Exercice 4

- Compléter le code ci dessous.
- Qu'affichera le code dans la console une fois complété ?

```go
type Animal interface {
  Crie()
  Mange()
}

type Chien struct {
  poids int
}
func (.....) Mange() {
  c.poids += 1
}
func (.....) Crie() {
  if c.poids < 5 {
    fmt.Println("OUAF")
  } else {
    fmt.Println("ouaf")
  }
}

type Chat struct {
  fatigue .....
}
func (.....) Mange() {
  c.fatigue = true
}
func (.....) Crie() {
  if c.fatigue {
    fmt.Println("...")
  } else {
    fmt.Println("miaou")
  }
}

func main() {
  animals := .....{
    &Chien{},
    &Chat{},
    &Chien{},
  }
  for _, animal := ..... {
    animal.Crie()
    animal.Mange()
    animal.Crie()
  }
}
```

Exercice 5

- Ajoutez une nouvelle espece d'animal au code de l'exercice 4, précisez les modifications que vous ferez afin que le code fonctionne toujours.
- Ajoutez à l'interface `Animal` la méthode `Vitesse() float32` qui calcule la vitesse de l'animal selon ses propriétés.
- Écrivez les modifications que vous apporterez à la fonction main pour tester cette nouvelle méthode.
- Que va afficher le code une fois toutes ces modifications effectuées ?

Correction

Exercice 0: QCM

Plusieurs réponses possibles, pas de points négatifs.

Le langage go est ?

a) Interprété.
b) **Compilé**.

Soit la fonction déclarée comme suit:

```
func f(a int) float32
```

a) Le type de retour de f est int
b) **Le type de retour de f est float32**
c) **La fonction f accepte un argument de type int**
d) La fonction f accepte un argument de type float

Soit la fonction déclarée comme suit:

```
func f(a float32) (float32, string)
```

a) Le type de retour de f est int
b) Le type de retour de f est float32
c) La fonction f accepte deux arguments de type int et string
d) **La fonction f renvoie deux valeurs de type int et float**

Lorsqu'on utilise plus un objet en go

a) Il faut libérer la mémoire manuellement.
b) La mémoire sera libérée à la fin de la fonction.
c) **La mémoire sera libérée quand l'objet ne sera plus accessible.**

En go, une structure

a) **Contient des attributs de types différents.**
b) Ne peut pas être passée par pointeur à une fonction.
c) **Peut être passée par copie à une fonction.**

Une méthode en go

a) Doit avoir au moins deux arguments.
b) **Ne peut être appelée que sur un objet.**
c) **Est définie par une syntaxe différente d'une fonction.**

Une Interface

a) Contient des attributs et des méthodes.
b) Ne peut pas être utilisée dans les listes
c) Sert à implémenter les `map`
d) **Est définie par une syntaxe différente d'une structure.**

Une slice

a) Ne peut pas changer de taille.
b) **Peut être passée par pointeur à une fonction.**
c) Peut être passée par copie à une fonction.

Exercice 1

a) Le code affiche:

```
8.5
14.5
```

b) Le code cherche l'indice de la valeur maximale dans le tableau, il affichera donc **3**.

c) On additionne les valeurs en utilisant une boucle **for**.

```
func main() {
    values := [5]float64{1.2, 2.3, 3.4, 4.5, 5.6}
    var sum float64
    for _, v := range values {
        sum += v
    }
    fmt.Printf("%.2f\n", sum)
}
```

Exercice 2

Complétez les types manquants dans ces fonctions.

Note: pour ce code, n'importe quel type entier serait accepté

```
func replaceEvens(arr []int, replacement int) {
  for i := 0; i < len(arr); i++ {
    if arr[i]%2 == 0 {
      arr[i] = replacement
    }
  }
}
```

```
func split(chaine []string, delim rune) []string {
  var result []string
  start := 0
  for i := 0; i < len(s); i++ {
    if string(s[i]) == delim {
      result = append(result, s[start:i])
      start = i + 1
    }
  }
  result = append(result, s[start:])
  return result
}
```

Exercice 3

La fonction **a** ne modifie pas de valeur.

La fonction **b** affecte la valeur de **i** à **j**.

La fonction **c** incrémente la valeur de **k** de 2.

Le code affiche donc 1, 1, 5.

Exercice 4

Une fois complété ainsi:

```
type Animal interface {
  Crie()
  Mange()
}
type Chien struct {
  poids int
}
func (c *Chien) Mange() {
  c.poids += 1
```

```
}
func (c *Chien) Crie() {
  if c.poids < 5 {
    fmt.Println("OUAF")
  } else {
    fmt.Println("ouaf")
  }
}

type Chat struct {
  fatigue bool
}
func (c *Chat) Mange() {
  c.fatigue = true
}
func (c *Chat) Crie() {
  if c.fatigue {
    fmt.Println("...")
  } else {
    fmt.Println("miaou")
  }
}

func main() {
  animals := []Animal{
    &Chien{},
    &Chat{},
    &Chien{},
  }
  for _, animal := range animals {
    animal.Crie()
    animal.Mange()
    animal.Crie()
  }
}
```

Le code affichera:

```
OUAF
OUAF
miaou
...
OUAF
OUAF
```

Exercice 5

a) On ajoute un crocodile qui grandit quand on lui donne à manger et crie plus longtemps si il est plus grand:

```
type Crocodile struct {
  longueur int
}
func (c *Crocodile) Mange() {
  c.longueur += 1
}
func (c *Crocodile) Crie() {
    fmt.Print("R")
    for i := 0; i < c.longueur + 2; i += 1 {
        fmt.Print("A")
    }
```

```
    fmt.Println("h")
}
```

Et on modifie le main pour l'ajouter à la liste.

```
animals := []Animal{
&Chien{},
&Chat{},
&Chien{},
&Crocodile{},
}
```

b) On ajoute la fonction Vitesse à l'interface:

```
type Animal interface {
  Crie()
  Mange()
  Vitesse() float32
}
```

On ajoute ensuite les trois méthodes Vitesse:

```
func (c *Chien) Vitesse() float32 {
// on suppose qu'un chien plus lourd court moins vite
  return 20.0 - float32(c.poids)
}

func (c *Chat) Vitesse() float32 {
  if c.fatigue {
      return 35.0
  } else {
    return 50.0
  }
}
func (c *Crocodile) Vitesse() float32 {
  return 12.0 * float32(c.longueur)
}
```

c) On modifie la boucle du main pour tester cette fonction:

```
  for _, animal := range animals {
    animal.Crie()
    fmt.Println("Vitesse: %f", animal.Vitesse())
    animal.Mange()
    animal.Crie()
    fmt.Println("Vitesse: %f", animal.Vitesse())
  }
```