INF600A — Langages de script et langages dynamiques
Examen final (Hiver 2017)

Durée: 13h30 à 16h30 (≈)
Documentation : Documentation autorisée, dont tablette ou *laptop* — en «lecture»!

Nom: _____

Code permanent: | | | | | | | | | | | | |

1	2	3	4	5	6	Total
/12	/9	/10	/10	/9	/10	/60

- Répondez directement sur le questionnaire, en utilisant seulement le recto des pages.

Quelques petits rappels :

- `x.even?` : Détermine si un nombre `x` est pair — `x.even? == (x % 2 == 0)`.
- `"123".to_i == 123`
- `k1..k2` : `Range` **inclusif**, i.e., la borne supérieure `k2` est **incluse**.
  `[*k1..k2] == [k1, k1+1, ..., k2-1, k2]`
- `[*1..3].map { |x| [x] } == [[1], [2], [3]]`
  `[*1..3].flat_map { |x| [x] } == [1, 2, 3]`
- `c.include? x` : Détermine si `x` fait partie de la collection `c` — i.e., `x` est un des éléments de `c`, tels qu'énumérés par `each`.
- `c.all? { |x| pred(x) }` : Détermine si chacun des `x` de `c` satisfait le `prédicat`, e.g., `[10, 20].all? { |x| x.even? } == true (== [10, 20].all?(&:even?))`
- Les méthodes `puts` et `p` n'ont pas le même effet pour `String` et `Array` :

```
>> puts [10, 20]                >> p [10, 20]
10                              [10, 20]
20

>> puts ["10", "20"]            >> p ["10", "20"]
10                              ["10", "20"]
20
```

- «`refute` *`expr`*» est équivalent à «`assert !`*`expr`*».
- «`alias_method :synonyme, :nom_methode`» introduit un identificateur `synonyme` à `nom_methode` — donc, on a alors deux noms distincts pour appeler la même méthode.

1. Opérations de style fonctionnel du module `Enumerable` (12 pts)

[8] a) Soit l'extension suivante de la classe `Fixnum` (nombres entiers). Évaluez les expressions.

```
class Fixnum
  include Enumerable

  def each  # Énumère les chiffres du nombre -- de gauche à droite.
    self.to_s.each_char do |c|
      yield c.to_i
    end
  end
end
```

Exemples : `2017.to_a == [2, 0, 1, 7]; 2017.reduce(&:+) == 10`

```
# a.
2017.reject { |x| x }

# b.
2017.map { |x| "#{x}" * x }.join('-')


# c.
2017.map { |x| x.map { |y| 2 * y } }


# d.
2017.map { |x| x.include?(0) }


# e.
2017.select(&:even?).map { |x| [*0..x] }


# f.
2017.reject { |x| x > 2 }.flat_map { |x| [*0..x] }


# g.
2017.reduce(1) { |a, x| a * (x + 1) }


# h.
2017.reduce([[], []]) { |a, x|  (x < 5 ? a[0] : a[1]) << x; a }
```

[4] b) La méthode `any?` du module `Enumerable` détermine si **au moins un** des éléments d'une collection satisfait une condition, spécifiée par un bloc :

```
refute [].any? { |x| x >= 0 }
refute [11].any? { |x| x < 10 }
assert [0, 19, 30].any?(&:even?)
refute [1, 15, 33].any?(&:even?)
```

i. Donnez une mise en oeuvre de `any?` qui utilise `reduce`.

```
def any?




end
```

ii. Donnez une mise en oeuvre de `any?` qui utilise `each` et qui effectue une évaluation court-circuitée — i.e., qui retourne le résultat final **aussitôt que possible.**

[**Points bonus accordés** pour le traitement du cas spécial où le bloc est omis lors de l'appel, auquel cas on vérifie s'il y a au moins un élément **qui n'est pas faux (au sens Ruby)**.]

```
def any?




end
```

2. Ruby et `MiniTest` (9 pts)

Complétez les cas de test suivants, en `MiniTest`, pour qu'ils s'exécutent avec succès.

```
it "stub 1" do
  ch = 'abc'
  ch.stub :+, lambda { |x| x.empty? ? '!!' : x.reverse } do
    (ch + 'def').must_equal                                    # a.
    ('' + ch)   .must_equal                                    # b.
  end
  (ch + '').must_equal                                         # c.
end

class Foo
  def initialize( v ); @v = v; end
  def print; puts "#{@v}"; end
end

it "stub 2" do
  f1 = Foo.new(1)
  r = ''
  STDOUT.stub :puts, lambda { |l| r << l } do
    f1.print; Foo.new(99).print; f1.print
  end
  r.must_equal                                                 # d.
end

let(:m) { MiniTest::Mock.new }

it "mock 1" do
                                                               # e.

  m.plus(10).must_equal 0
  m.verify
end

it "mock 2" do
  m.expect( :foo, 'foo', [Fixnum, String] )
  m.expect( :bar, 'bar', ['foo'] )

                                                               # f.

  m.verify
end
```

3. Script `ga.rb` : Commande pour lister les cours inactifs (10 pts)

Dans le script `ga.rb` du devoir 2 (la version **qui n'utilise pas** `gli`!), on veut ajouter une commande `lister_inactifs` qui permet de **lister uniquement les cours inactifs**. La figure 1 présente des exemples comparant `lister` avec `lister_inactifs` : par défaut, le sigle et le titre du cours sont affichés tel qu'illustré, à moins qu'un `format` ne soit explicitement spécifié. Notez que le format par défaut utilisé pour `lister_inactifs` n'est pas le même que celui pour `lister`... et que c'est aussi la seule option!

```
$ ./ga.rb lister --avec_inactifs
INF1120 "Programmation I" ()
INF2120 "Programmation II" (INF1120)
INF3143? "Modelisation formelle" ()
INF3330? "Environnements de programmation" (INF2120)

$ ./ga.rb lister_inactifs
INF3143: 'Modelisation formelle'
INF3330: 'Environnements de programmation'

$ ./ga.rb lister_inactifs --format='%S [%T] (%C)'
INF3143 [Modelisation formelle] (3)
INF3330 [Environnements de programmation] (3)
```

Figure 1: Exemples d'exécution de la commande `lister_inactifs`.

Complétez le code de la fonction `lister_inactifs` à la page 6.

Quelques contraintes à respecter :

- **Vous devez utiliser le style fonctionnel**.
- La liste des cours inactifs doit être ordonnée selon le sigle — voir les exemples.
- Comme dans le devoir 2, l'affichage ne doit pas être fait dans cette méthode!

Rappels des méthodes de la classe `Cours`

```
class Cours
  include Comparable
  attr_reader :sigle, :titre, :nb_credits, :prealables

  def initialize( sigle, titre, nb_credits, *prealables, actif: true )
  def to_s( le_format = nil, separateur_prealables = ... )
  def <=>( autre )
  def desactiver
  def activer
  def actif?
end
```

```
def lister_inactifs( les_cours )

end
```

4. Application `mini-sed` et le *gem* `gli` (10 pts)

Dans l'application `mini-sed` (laboratoire #8), on veut ajouter une commande `insert`, semblable à celle de `sed`, qui reçoit en argument un `motif` et une `chaine` et qui **insère** une ligne contenant la `chaine` (+ saut de ligne) **avant** chaque ligne qui matche le `motif`.

```
$ cat foo.txt
abc
def
axayaz

$ bin/mini-sed insert a 1234 foo.txt
1234
abc
def
1234
axayaz

$ bin/mini-sed insert . 1234 foo.txt    # Motif = '.'
1234
abc
1234
def
1234
axayaz

$ bin/mini-sed insert 999 1234 foo.txt
abc
def
axayaz
```

Donnez une mise en oeuvre de cette commande pour `mini-sed`.
Vous devez spécifier (p. 9) les deux éléments de cette mise en oeuvre :

a. La spécification de la commande au niveau de l'IPM dans `bin/mini-sed`.

b. La mise en oeuvre de la fonctionnalité dans `lib/mini-sed/mini-sed.rb`. Comme dans le labo #8, la méthode doit être réalisée avec un `TransformateurDeFlux`, pour assurer un traitement incrémental des données et du résultat.

Note : La page suivante présente des extraits de `bin/mini-sed` et `lib/mini-sed/mini-sed.rb` — la variable `tdf` est une abréviation pour `TransformateurDeFlux`.

```
# Une des méthodes définies dans le fichier bin/mini-sed.
def commande_sans_option commande, nb_arguments: 1
  command commande do |c|
    c.action do |global_options,options,args|
      les_args, fichiers = args[0...nb_arguments], args[nb_arguments..-1]
      fichiers = [:stdin] if fichiers.empty?

      fichiers.each do |fichier|
        MiniSed.traiter_fichier( fichier, global_options[:"in-place"] ) do |flux|
          MiniSed.send commande, flux, *les_args
        end
      end
    end
  end
end
```

```
# Deux des méthodes définies dans le fichier lib/mini-sed/mini-sed.rb.
  def self.substitute( flux, motif, chaine, global )
    tdf = TransformateurDeFlux.new( flux, motif, remplacement: chaine, global: global )

    def tdf.each
      global = @options[:global]; chaine = @options[:remplacement]; motif = /#{@motif}/
      @flux.each do |ligne|
        if motif =~ ligne
          yield( global ? ligne.gsub!(motif, chaine) : ligne.sub!(motif, chaine) )
        else
          yield( ligne )
        end
      end
    end

    tdf
  end

  def self.print( flux, motif )
    tdf = TransformateurDeFlux.new( flux, motif )

    def tdf.each
      @flux.each do |ligne|
        yield(ligne) if /#{@motif}/ =~ ligne
        yield(ligne)
      end
    end

    tdf
  end
```

a. Ajout à faire dans `bin/mini-sed` :

b. Ajout à faire dans `lib/mini-sed/mini-sed.rb` :

5. Blocs, `yield` et métaprogrammation (9 pts)

Soit la classe `Foo` définie ci-dessous. Indiquez ce qui sera imprimé par chacun des appels à la méthode `p`, et ce **dans l'ordre indiqué**. (Voir les **rappels** p. 1 pour la méthode `p`.)

```ruby
class Foo
  def self.foo( x )
    return x unless block_given?

    yield( x + 1 )
  end

  def foo( x )
    Foo.foo( yield(x) )
  end
end
```

```ruby
class Foo
  def self.refoo( o )
    def o.foo( x = nil )
      x ? x : yield(@foo)
    end

    def o.foo=( x )
      @foo = x
    end
  end
end
```

```ruby
# a.
p Foo.send( "f" + "o" * 2, 99 )


# b.
p Foo.send( :foo, 9 ) { |x| 10 * x }


# c.
p (f1 = Foo.new).foo(3) { |x| 2 * x }


# d.
Foo.refoo(f1)
f1.foo = 10
p f1.foo { |x| 3 * x }


# e.
a = [:'self.foo=', "9 + @foo * "]
p a.map { |m| f1.instance_eval "#{m} 2" }


# f.
f1.foo = "foo"
p f1.send( f1.instance_eval("@foo") ) { |x| x + x }
```

6. API coulantes (DSL interne) pour décrire des ordinateurs (10 pts)

Le listing Ruby aux pages 13 et 14 présente des extensions des classes `Object` et `Fixnum` ainsi que des classes `Processeur` et `Ordinateur`. Les méthodes de ces classes peuvent être utilisées pour décrire des ordinateurs, et ce avec différents styles d'API coulantes.

a. Un *cluster* — grappe de serveurs ou grappe de calcul — est une machine composée de deux ou plusieurs ordinateurs. Les deux méthodes ci-bas reçoivent un argument, `cluster`, qui est un `Array` d'`Ordinateur`s. Décrivez en quelques mots ce que fait chaque méthode et donnez-lui un nom **plus significatif**.

```ruby
# Cette méthode...
#
#
#
#
def ______________________________( cluster )
  cluster
    .map(&:numero_serie)
    .sort { |x, y| y <=> x }
end


# Cette méthode...
#
#
#
#
def ______________________________( cluster )
  cluster
    .flat_map(&:processeurs)
    .map(&:nb_coeurs)
    .reduce(0, :+)
end
```

b. Pour chacune des séries d'appels à la page 12, **indiquez ce qui sera imprimé**.

   Si une erreur est signalée :

   (i) Indiquez quelle serait cette erreur ;
   (ii) Modifiez **la description** pour qu'elle devienne valide ;
   (iii) Indiquez ce qui serait imprimé suite à votre modification.

Note : Pour faciliter la lecture, vous pouvez détacher les deux dernières pages (p. 13–14).

```ruby
# a.
p0 = Processeur.new.nb_coeurs(4).sorte(:CPU)
puts p0.sorte


# b.
p1 = Processeur.new(:FPGA, 128)
puts p1.nb_coeurs


# c.
o0 = Ordinateur.create(100) do |o|
  o.processeur = Processeur.new.nb_coeurs(4)
end
puts o0.processeur(0).nb_coeurs


# d.
o1 = Ordinateur.create(101) do |o|
  o.processeur = Processeur.new(:CPU, 2)
  o.processeur = Processeur.new(:GPU, 128)
end
puts o1.processeur(0)


# e.
o2 = Ordinateur.create(102) do |o|
  o.processeurs << Processeur.new
                             .nb_coeurs(64)
                             .sorte(:GPU)
end
puts o2


# f.
o3 = Ordinateur.new(103) do |o|
  o.processeur = processeur(:GPU).avec(64.coeurs)
  o.processeur = processeur(:CPU).avec(2.coeurs)
end
puts o3
```

```ruby
class Object
  def processeur( sorte )
    Processeur.new( sorte )
  end
end

class Fixnum
  def coeurs
    self
  end
end

class Processeur
  def initialize( sorte = nil, nb_coeurs = nil )
    @sorte, @nb_coeurs = sorte, nb_coeurs
  end

  def sorte( sorte = nil )
    return @sorte unless sorte

    @sorte = sorte
    self
  end

  def nb_coeurs( nb_coeurs = nil )
    return @nb_coeurs unless nb_coeurs

    @nb_coeurs = nb_coeurs
    self
  end
  alias_method :avec, :nb_coeurs

  def valide?
    [:CPU, :GPU].include?(@sorte) && @nb_coeurs && @nb_coeurs >= 1
  end

  def to_s
    fail "to_s: Processeur invalide" unless valide?
    "#{@sorte}/#{@nb_coeurs}"
  end
end
```

```ruby
class Ordinateur
  attr_reader :numero_serie

  def initialize( numero_serie = nil )
    @numero_serie = numero_serie
    @processeurs = []
  end
  private_class_method :new

  def self.create( numero_serie = nil )
    nouveau = new numero_serie
    yield nouveau
    fail "create: Ordinateur invalide" unless nouveau.valide?
    nouveau
  end

  def numero_serie=( n )
    @numero_serie = n
  end

  def processeur( p )
    return @processeurs[p] if p.kind_of?(Fixnum)
    @processeurs << p
  end

  def processeur=( p )
    @processeurs << p
  end

  def processeurs( *p )
    @processeurs += p
  end

  def valide?
    @numero_serie &&
      !@processeurs.empty? && @processeurs.all?(&:valide?)
  end

  def to_s
    fail "to_s: Ordinateur invalide" unless valide?
    "#<Ordi#{@numero_serie}: #{@processeurs.join('; ')}>"
  end
end
```