INF600A — Langages de script et langages dynamiques
Examen final (Hiver 2016)

Durée: 9h30 – 12h30
Documentation : Documentation personnelle, y compris *laptop* — en mode «lecture»!

Nom: _____

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

1	2	3	4	5	6	Total
/13	/6	/6	/10	/5	/10	/50

- Répondez directement sur le questionnaire, en utilisant seulement le recto des pages — les versos sont pour vos brouillons et ne seront pas corrigés!
- L'examen comporte six (6) questions pour un total sur 50.

  La question 5, sur la métaprogrammation, compte 5 items qui seront évalués à **2 points chacun**, mais avec un total sur 5. Il est possible d'obtenir des **points bonus** pour cette question.
- Quelques petits rappels :
  - `x.even?` : Détermine si un nombre `x` est pair — `x.even? == (x % 2 == 0)`.
  - `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`.
  - Les méthodes `puts` et `p` n'ont pas le même effet pour `String` et `Array` :

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

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

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

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

  - Si `a` et `b` sont des `Fixnum`s (entiers), alors «`/`» dénote la partie entière et «`%`» le reste (le modulo). Exemple : `7 / 3 == 2` et `7 % 3 == 1`.
  - `1..n` dénote un `Range` avec une borne **inclusive** — la borne supérieure est donc incluse dans le `Range`.

1. Opérations, de style fonctionnel, du module `Enumerable` (13 pts)

[8] a) Soit l'extension suivante de la classe `Fixnum` (nombres entiers) :

```
class Fixnum
  include Enumerable

  def each # Enumere les chiffres du nombre, de droite a gauche.
    (yield(0); return) if self == 0
    v = self
    while v != 0; yield(v % 10); v /= 10; end
  end
end
```

Exemple d'utilisation : `123.map { |x| x } == [3, 2, 1]`

Évaluez alors les expressions suivantes :

```
# a.
8703.select { |x| x >= 4 }


# b.
8703.reject { |x| x.even? }


# c.
8703.reject { |x| x }


# d.
8703.map { |x| x * x }


# e.
8703.map { |x| x.include? 3 }


# f.
8703.map { |x| x.map { |y| y + 1 } }


# g.
8703.reduce(0) { |a, x| a + 2 * x }


# h.
8703.reduce([[], []]) { |a, x| (x.even? ? a[0] : a[1]) << x; a }
```

[5] b) Une des méthodes du module `Enumerable` est la méthode `all?`, qui détermine si **tous** les éléments d'une collection satisfont une certaine condition, spécifiée par un bloc :

```
assert [].all? { |x| x >= 0 }
refute [11].all? { |x| x < 10 }
assert [0, 12, 30].all? { |x| x.even? }
refute [0, 15, 30].all? { |x| x.even? }
```

i. Donnez une mise en oeuvre de `all?` en utilisant `reduce`.

```
def all?




end
```

ii. Donnez une mise en oeuvre de `all?` en utilisant `each`, mise en oeuvre qui doit effectuer une évaluation court-circuitée — i.e., qui termine aussitôt que le résultat final est connu de façon certaine.

```
def all?




end
```

2. Tests unitaires avec *mocks* et *stubs* (6 pts)

Le code ci-bas présente deux classes, `Alarme` et `Alerte`, dont les fonctionnalités sont brièvement décrites dans les commentaires. La classe `Alarme` pourra être utilisée par `Alerte` par *injection de dépendances* (via *Constructor injection*) lors de la construction du vrai logiciel. Évidemment, durant les tests, on ne veut pas déclencher une vraie alarme, d'où la nécessité d'utiliser un objet `mock` approprié. Complétez les tests unitaires aux pages suivantes, pour assurer que le comportement désiré pour `Alerte` soit satisfait.

Rappel : Les instructions `let` sont réévaluées de façon indépendante **pour chaque cas de test**, et ce avant l'exécution de chaque `before` et chaque `it`.

```
# Gere l'interface (physique) avec une alarme.
class Alarme
  def allumer
    # ... actions pour faire sonner l'alarme ...
  end

  def eteindre
    # ... actions pour eteindre l'alarme lorsqu'allumee ...
  end
end

# Gere les alertes.
# - Allume et eteint l'alarme associee.
# - Prend note de l'heure de debut et fin de chaque alerte.
class Alerte
  attr_reader :activations, :desactivations

  def initialize( alarme )
    @alarme = alarme
    @activations = []
    @desactivations = []
  end

  def activer
    @alarme.allumer
    @activations << Time.now
  end

  def desactiver
    @alarme.eteindre
    @desactivations << Time.now
  end
end
```

```
let(:alarme_mock) { MiniTest::Mock.new }
let(:alerte) { Alerte.new alarme_mock }

describe "#activer" do
  before do
    # A COMPLETER!









  end

  it "prend note de l'activation et fait sonner l'alarme" do
    alerte.activations.size.must_equal 1
    alarme_mock.verify
  end
end

describe "#activer puis #desactiver" do
  before do
    # A COMPLETER!









  end

  it "prend note de l'activation/desactivation et eteint
      l'alarme apres l'avoir allumee" do
    alerte.activations.size.must_equal 1
    alerte.desactivations.size.must_equal 1
    alarme_mock.verify
  end
end
```

```
describe "temps des alertes" do
  let(:heure_activation) { Time.new( 2015, 9, 22, 9, 0, 0 ) }

  before do
    # A COMPLETER!









  end

  it "prend note de l'heure d'activation et allume l'alarme" do
    alerte.activations.must_equal [heure_activation]
    alarme_mock.verify
  end
end
```

3. Script `biblio.rb` : Commande pour lister les livres perdus (6 pts)

Dans le script `biblio.rb` du devoir 2, on veut ajouter une nouvelle commande, qui permet **de lister uniquement les livres perdus**, en indiquant le titre et l'emprunteur du livre.

La figure 1 présente des exemples d'exécution de cette commande.

Complétez le code de la fonction `lister_perdus` — page 8.

Quelques contraintes à respecter :

- **Vous devez utiliser le style fonctionnel** de programmation.
- La liste des livres perdus doit être ordonnée selon le titre — voir l'exemple.
- Si approprié (?), vous pouvez utiliser la fonction suivante, elle aussi déjà définie :

```
def emprunt_avec_titre( emprunts, titre )
  emps = emprunts.select { |emprunt| emprunt.titre == titre }

  erreur "Plusieurs emprunts avec titre '#{titre}'" if emps.size > 1

  emps.first
end
```

```
$ ./biblio.rb lister
Guy Tremblay :: [ Hunt et Thomas ] "Programming Ruby"
Joe Bidon :: [ Hunt et Thomas ] "The Pragmatic Programmer"

$ ./biblio.rb indiquer_perte "Programming Ruby"

$ ./biblio.rb indiquer_perte "The Pragmatic Programmer"

$ ./biblio.rb lister

$ ./biblio.rb lister_perdus
'Programming Ruby' => Guy Tremblay
'The Pragmatic Programmer' => Joe Bidon
```

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

```
def lister_perdus( les_emprunts )

end
```

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

Dans l'application `mini-sed` (laboratoire # 7), on veut ajouter une nouvelle commande nommée «`change`». Cette commande reçoit un motif et une chaine et, si l'option `quiet` n'est pas indiquée, chaque ligne qui matche le motif est **remplacée** au complet par une ligne qui ne contient que la chaine indiquée :

```
$ cat foo.txt
abc
def
zzabxxx
xxxx

$ bin/mini-sed change a 1234 < foo.txt
1234
def
1234
xxxx

$ bin/mini-sed change . . < foo.txt
.
.
.
.


$ bin/mini-sed -q change a 1234 < foo.txt

$
```

Donnez une mise en oeuvre de cette nouvelle commande pour `mini-sed`.
Vous devez spécifier — page 11 — les deux composantes de cette mise en oeuvre :

a. La spécification de la commande au niveau de l'IPM, dans le fichier du programme principal `bin/mini-sed`.

b. La mise en oeuvre des fonctionnalités de la commande dans le fichier `lib/mini-sed/mini-sed.rb`, mise en oeuvre qui doit être réalisée **dans un style fonctionnel**.

Voir à la page suivante (page 10) pour des extraits du fichier `bin/mini-sed` et du fichier `lib/mini-sed/mini-sed.rb` — notez la solution pour `delete`, différente de celle présentée en classe, qui traite directement le cas (spécial) où `quiet` est `true`.

```ruby
# Methode definie 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,
                       global_options[:quiet],
                       *les_args,
                       flux.readlines
        end
      end
    end
  end
end
```

```ruby
# Deux des methodes definies dans le fichier lib/mini-sed/mini-sed.rb
  def self.delete( quiet, motif, lignes )
    return [] if quiet

    lignes.reject { |ligne| /#{motif}/ =~ ligne }
  end


  def self.print( quiet, motif, lignes )
    res = []
    lignes.each do |ligne|
      res << ligne if /#{motif}/ =~ ligne
      res << ligne unless quiet
    end

    res
  end
```

a. Ajout au fichier `bin/mini-sed`

b. Ajout au fichier `lib/mini-sed/mini-sed.rb`

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

Soit la classe `UneClasse` 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 `p`!).

```ruby
class UneClasse
  def self.bis( x )
    yield(x) + yield(x)
  end

  def bis( x )
    yield( yield(x) )
  end
end
```

```ruby
class UneClasse
  def self.ms( *symbs )
    symbs.each do |s|
      define_method s do
        instance_eval "@#{s}"
      end

      define_method "#{s}=" do |v|
        instance_eval "@#{s} = '#{s}-#{v}'"
      end
    end
  end
end
```

```ruby
# a.
p UneClasse.bis(2) { |x| x * x }
```

```ruby
# b.
UneClasse.class_eval "def self.bis( x ) yield(x) * yield(x); end"
p UneClasse.bis(2) { |x| x * x }
```

```ruby
# c.
c1, c2 = (1..2).map { UneClasse.new }
p c1.bis(3) { |x| x * x }
```

```ruby
# d.
UneClasse.ms( :foo, :bar, :bazz )
c1.foo = 10; c1.bar = 10; c1.bazz = 10
p [:foo, :bar].map { |m| c1.send m }
```

```ruby
# e.
p [:foo, :bazz].map { |o| c2.instance_eval "@#{o}= 33"; c2.send o }
```

6. APIs coulantes (DSLs internes) pour décrire des cours (10 pts)

Le listing Ruby aux pages 15 et 16 présente une classe `Cours` comportant diverses méthodes. Ces méthodes peuvent être utilisées pour décrire des cours, et ce en utilisant deux des approches vues en cours : *i*) avec chainage de méthodes et *ii*) avec constructeur et bloc.

Pour chacune des séries d'appels à la page 14, **indiquez (du coté droit) ce qui sera imprimé**.

Si une erreur est signalée, **indiquez quelle serait cette erreur** (en indiquant le message exact si généré par un `fail` dans la classe `Cours`). Toujours en cas d'erreur, si possible, **modifiez le code de la description du cours pour qu'elle devienne correcte et produise une description de cours valide**.

Note : Pour faciliter la lecture du code, vous pouvez détacher les deux (2) dernières pages (pages 15–16).

```ruby
# a.
inf1120 = Cours.new
  .sigle( :INF1120 )
  .titre( "Prog. I" )
puts inf1120



# b.
inf2120 = Cours.new
  .titre( "Prog. II" )
  .prealable( inf1120 )
  .fin
puts inf2120



# c.
inf3105 = Cours.new
  .sigle( :INF3105 )
  .titre( "Str. de don." )
  .ajouter_prealable( inf1120 )
  .ajouter_prealable( inf2120 )
  .fin
puts inf3105



# d.
inf1130 = Cours.creer do |c|
  c.sigle = :INF1130
  c.titre "Math. info."
end
puts inf1130



# e.
inf3105 = Cours.creer :INF3105 do |c|
  c.titre = "Str. de don."
  c.prealable inf1130
  c.prealables << inf2120
end
puts inf3105
```

```ruby
class Cours
  attr_writer :sigle, :titre
  attr_accessor :prealables

  def initialize( sigle = nil )
    @sigle = sigle
    @prealables = []
  end

  def self.creer( sigle = nil )
    nouveau_cours = new sigle

    yield nouveau_cours

    unless nouveau_cours.valide?
      fail "*** Dans creer: Cours mal construit"
    end

    nouveau_cours
  end

  def sigle( s = nil )
    return @sigle unless s
    @sigle = s
    self
  end

  def titre( t = nil )
    return @titre unless t
    @titre = t
    self
  end
```

```
  def prealable( p )
    @prealables << p
    self
  end

  def ajouter_prealable( prealable )
    prealables << prealable
  end

  def fin
    fail "*** Dans fin: Cours mal construit" unless valide?
    self
  end

  def to_s
    fail "*** Dans to_s: Cours mal construit" unless valide?

    unless prealables.empty?
      les_prealables = ": "
      les_prealables << prealables
                          .map { |c| c.sigle.to_s }
                          .join( ", " )
    end

    "#<#{sigle} '#{titre}'#{les_prealables}>"
  end

  def valide?
    sigle && titre && prealables
  end
end
```