INF600A — Langages de script et langages dynamiques
Examen final (Automne 2018)

Durée: 13h30 à 16h30
Documentation personnelle permise, y compris *laptop* ou tablette, mais en mode «lecture de documents personnels» — pas de sites Web!

Nom: _____

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

1	2	3	4	5	6	Total
/12	/8	/8	/10	/5	/7	/50

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

Quelques petits rappels :

- `foo(&:bar) == foo { |x| x.bar }`
- `'123'.to_i => 123; 'abc'.to_sym => :abc; '1 X-3'.to_sym => :'1 X-3'`
- `/abc/ =~ 'abc' => 0; /abc/ =~ 'xyabc' => 2; /abc/ =~ 'xy' => nil`
- `{:a => 10, :b => 20, :c => 30}.keys   => [:a, :b, :c]`
  `{:a => 10, :b => 20, :c => 30}.values => [10, 20, 30]`
- `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 *pred*icat, e.g., `[0, 10].all?(&:even?) => true`; `[0, 10].all?(&:zero?) => false`.
- `irb` affiche en utilisant `p`, via `inspect`. Par contre, `puts` affiche via `to_s`.

```
>> [10, 'abc']
=> [10, "abc"]

>> nil
=> nil

>> o = Object.new
=> #<Object:0x000000023ce838>

>> o.instance_eval "@x = 12"
=> 12

>> o
=> #<Object:0x000000023ce838 @x=12>
```

```
$ cat foo.rb
puts nil
puts [10, 'abc']

o = Object.new
o.instance_eval "def to_s; 'Foo' end"
p o
puts o

$ ruby foo.rb

1
abc
#<Object:0x00000001f9d110>
Foo
```

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

[8] a) Soit l'extension suivante de la classe `Symbol`.

```
class Symbol
  include Enumerable

  def each  # Énumère les caractères du symbole -- de gauche à droite.
    self.to_s.each_char { |c| yield(c) }
  end
end
```

Exemples : `:ab.map { |x| x } => ['a', 'b']; :ab.reduce(&:+).to_sym => :ab`

Évaluez les expressions ci-bas — comme dans `irb` donc affichées avec `p`.

```
# a.
>> :x209.select { |x| x }
=>

# b.
>> :x209.reject { |x| /[^1-9]/ =~ x }.map(&:to_i).reduce(:*)
=>

# c.
>> :foo.flat_map { |x| :"#{x}".map { |y| y * 3 } }
=>

# d.
>> :bar.map { |x| !x.include?('a') } # Notez le <<!>>
=>

# e.
>> :'2147'.map(&:to_i).select(&:even?).map{ |x| [x, x*2] }
=>

# f.
>> :bar.reduce('') { |a, x|  x + a + x }
=>

# g.
>> :'142'.map(&:to_i).reduce(0) { |a, x|  a + (x <=> a) }
=>

# h.
>> :bar.group_by { |x| /[aeiou]/ =~ x }
=>
```

[4] b) La méthode `count` du module `Enumerable` détermine le nombre d'éléments d'une collection qui satisfont une condition, spécifiée par un bloc :

```
assert [10, 11, 12, 13, 14].count { |x| x <= 10 } == 1
assert [10, 11, 12, 13, 14].count(&:even?) == 3
```

Vous devez écrire deux mises en œuvre de `count` respectant les conditions indiquées ci-bas — **sans utiliser d'autres méthodes d'**`Enumerable` (sauf évidemment `reduce` pour i.). {**Bonus** : **aucun** bloc `=>` nombre d'éléments, par ex., `[3, 1].count => 2`.}

i. Donnez une mise en oeuvre de `count` qui utilise `reduce` — style **fonctionnel**.

```
def count




end
```

ii. Donnez une mise en oeuvre de `count` qui utilise `each` — style impératif.

```
def count




end
```

2. Script `gv.rb` : Les vins les mieux notés (8 pts)

Avec le script `gv.rb` du devoir #2, on veut pouvoir identifier **le ou les vins qui sont les mieux notés** — c'est-à-dire le ou les vins qui ont été bus et qui ont reçu **la note la plus élevée**. La figure 1 présente des exemples d'exécution.

```
$ ./gv.rb lister
1 [rouge - 32.65$]: Chianti 2014, Volpaia (10/06/18) => 4 {Tres bon.}
2 [rouge - 30.35$]: Barolo 2012, Fontanafreda (10/06/18) => 4 {Excellent.}
4 [blanc - 16.50$]: Alsace 2016, Pfaff (03/07/18) => 3 {Aromatique.}
6 [rose - 18.00$]: Languedoc 2016, Clavel (03/07/18) =>  {}
7 [blanc - 12.50$]: Aligote 2017,  (22/09/18) => 2 {Frais.}

$ ./gv.rb mieux_notes | ./gv.rb - lister
2 [rouge - 30.35$]: Barolo 2012, Fontanafreda (10/06/18) => 4 {Excellent.}

$ ./gv.rb mieux_notes -3 | ./gv.rb - lister
2 [rouge - 30.35$]: Barolo 2012, Fontanafreda (10/06/18) => 4 {Excellent.}
1 [rouge - 32.65$]: Chianti 2014, Volpaia (10/06/18) => 4 {Tres bon.}
```

Figure 1: Exemples d'exécution pour identifier les vins les mieux notés.

Complétez le code de la méthode `mieux_notes` (p. 6) en utilisant **le style fonctionnel** — donc avec les méthodes du module `Enumerable`, et sans utiliser `each` — et en utilisant le *pattern-matching* avec groupe et capture lorsqu'approprié. De plus :

- Lorsque l'option «`-k`» n'est pas spécifiée, cela est équivalent à «`-1`».
- Seuls les vins ayant **exactement la meilleure note** sont indiqués. Ainsi, dans les exemples, seuls ceux avec une note de 4 (la note maximale) sont indiqués. Il est donc possible qu'il y ait **moins** que `k` vins sélectionnés.
- Les vins avec la meilleure note sont présentés **en ordre croissant de prix** — donc du moins cher au plus cher. C'est pourquoi dans l'exemple où un seul vin est indiqué, il s'agit de celui qui est le moins cher.
- Comme pour `selectionner` et `trier`, la sortie produite (sur `stdout`) est une collection d'enregistrements.
- Les cas d'erreurs suivants doivent être traités et signalés :
  - Nombre invalide pour `k`, par exemple, `-0` ou `-2ab`.
  - Argument en trop, par exemple, «`./gv.rb mieux_notes -3 foo`».

Autres contraintes à respecter et suggestions :

- La méthode `mieux_notes` est appelée du programme principal (`gv.rb`) comme toutes les autres méthodes du devoir #2, donc avec la collection `les_vins` en argument. Quant aux arguments et options fournis sur la ligne de commande, ils sont dans `ARGV` — sans l'option pour spécifier le dépôt («`--depot=...`» ou «`-`») qui a déjà été traitée.
- Si nécessaire, utilisez les méthodes suivantes :

```
# Émet le message indiqué sur STDERR puis termine l'exécution avec exit.
def erreur( msg ); ... end

# Vérifie que args est vide: si non vide alors appelle erreur (=> exit).
def verifier_arguments_en_trop( args ); ... end
```

- La méthode `Enumerable#sort` peut être appelée sans argument ou avec un bloc qui indique comment comparer les éléments, bloc qui doit retourner `-1, 0, 1` (comme `<=>`). Si aucun bloc n'est spécifié, la méthode `<=>` appropriée est utilisée.

```
>> [3, 2, 4, 1].sort
=> [1, 2, 3, 4]

>> [3, 2, 4, 1].sort { |x, y| y <=> x }
=> [4, 3, 2, 1]

>> [3, 2, 4, 1].sort { |x, y| x % 2 <=> y % 2 }
=> [2, 4, 3, 1]

>> [[4, 1], [2, 3], [1, 8]].sort { |x, y| x.first <=> y.first }
=> [[1, 8], [2, 3], [4, 1]]
```

- La méthode `Enumerable#take(n)` sélectionne les `n` premiers éléments — ou moins si la collection en contient moins que `n` :

```
>> [3, 2, 4, 1].take(0)
=> []

>> [3, 2, 4, 1].take(2)
=> [3, 2]

>> [3, 2, 4, 1].take(10)
=> [3, 2, 4, 1]
```

```
#===========
# Commande mieux_notes
#
# Selectionne les k vins qui sont les mieux notes
# (k = 1 si l'option -k n'est pas specifiee)
#
# Arguments: [-k]
#
# Erreurs:
# - k pas un nombre valide (doit etre un entier >= 1)
# - argument(s) en trop
#===========

def mieux_notes( les_vins )
    # A COMPLETER




end
```

3. Application `minised` et le *gem* `gli` (8 pts)

Dans l'application (le *gem*) `minised` (diapos & labo. #6), on veut ajouter une commande `read`, semblable à celle de `sed`, qui reçoit un `motif` et le nom d'un `fichier_a_inserer` et qui **insère** le contenu (les lignes) du fichier **après** chaque ligne qui matche `motif`. Des exemples sont présentés ci-bas — dans l'appel à `--help`, le «\» n'est pas significatif (ligne trop longue pour la largeur de la page).

```
$ cat foo.txt
abc
def
ab ab xx yy

$ cat lignes.txt
123
===

$ minised read --help
NAME
    read - Insere les lignes de fichier_a_inserer apres chaque ligne\
           qui matche motif

SYNOPSIS
    minised [global options] read motif fichier_a_inserer [fichier...]

$ minised read "ab" lignes.txt <foo.txt
abc
123
===
def
ab ab xx yy
123
===

$ minised read "^[^a]" lignes.txt foo.txt
abc
def
123
===
ab ab xx yy
```

Vous devez spécifier (p. 9) les deux éléments pour mettre en œuvre cette commande `read` :

a. La spécification de la commande au niveau de l'IPM dans `bin/minised`, notamment **pour qu'elle produise l'information indiquée par** `--help`.

   Remarque : pour rendre le code moins répétitif (*DRY*), une nouvelle méthode `commande_minised` a été définie dans `bin/minised` : voir plus bas. Elle remplace la méthode vue dans les diapos (après le labo #6 : `commande_sans_option`), puisqu'elle peut définir tant une commande sans option locale qu'une commande avec une (1) option locale (de type *switch*). **Vous devez utiliser cette méthode!**

b. La mise en œuvre de la fonctionnalité (traitement des flux : `#each` pour le flux d'entrée vs. `#<<` pour le flux de sortie) dans `lib/minised/minised.rb`.

Note : `IO.readlines(nom_de_fichier)` lit le contenu du fichier et retourne un `Array` des lignes lues (une ligne = une `String`). Pour simplifier, vous pouvez supposer que `fichier_a_inserer` **existe** — donc pas besoin de traiter le cas où le fichier n'existe pas.

```
# La méthode commande_minised, définie dans bin/minised, que vous devez utiliser.

def commande_minised( nom_cmd,
                      nb_arguments: 1,
                      switch: nil, desc_switch: nil )
  DBC.require (switch && desc_switch) || (switch.nil? && desc_switch.nil?)

  command nom_cmd do |c|
    (c.desc desc_switch; c.switch switch) if switch

    c.action do |global_options,options,args|
      les_args = args.shift(nb_arguments)
      fichiers = args.empty? ? [:stdin] : args
      in_pl = global_options[:'in-place']

      fichiers.each do |fich|
        MiniSed.traiter_fichier(fich, in_place: in_pl) do |f_in, f_out|
          args_send = [nom_cmd, f_in, f_out, *les_args]
          args_send << options[switch] if switch
          MiniSed.send( *args_send )
        end
      end
    end
  end
end
```

---

a. Ajout à faire dans `bin/minised` pour décrire la commande `read` :

b. Ajout à faire dans `lib/minised/minised.rb` pour mettre en œuvre le traitement des flux requis pour `read` — vous pouvez supposer que le fichier à insérer existe :

4. Blocs, `yield` et métaprogrammation (10 pts)

[6] a) Soit la classe `Bar` définie ci-dessous. Évaluez les expressions ci-bas — comme dans `irb` donc affichées avec `p` — et ce **dans l'ordre indiqué**.

```ruby
class Bar
  def self.bar( x )
    if block_given?
      yield( 10 * x )
    else
      x
    end
  end

  def bar( x )
    yield( yield(x) )
  end
end
```

```ruby
class Bar
  def self.define( n )
    define_method n do |x|
      x.even? ? x : 2 * x
    end
  end
end
```

```ruby
# a.
>> Bar.send( ("r" + "a" + "b").reverse, 98 )
=>


# b.
>> Bar.send( :bar, 3 ) { |x| x + 1 }
=>


# c.
>> (Bar.new.class).send( :bar, 99, &:even? )
=>


# d.
>> (b1 = Bar.new).bar(5) { |x| 3 * x }
=>


# e.
>> Bar.define(:bar); b1.bar(9)
=>


# f. ...en supposant que e. a ete execute au prealable!
>> [:bar, "bar"].map { |m| b1.instance_eval { send(m,  5) } }
=>
```

[4] b) Soit la classe `SClass` ci-bas. Indiquez ce qui sera imprimé par les appels à `p` ou `puts` ci-dessous — dans l'ordre indiqué. (Voir p. 1 pour un bref rappel sur `p` vs. `puts`.)

```ruby
class SClass
  def self.new( *champs )
    klass = Class.new

    klass.define_singleton_method(:[]) do |*vals|
      tc = klass.new
      hash, k = {}, 0
      champs.each do |c|
        hash[c], k = vals[k], k+1
      end
      tc.instance_variable_set "@hash", hash

      tc
    end

    champs.each do |c|
      klass.send(:define_method, c) { @hash[c] }
    end

    klass.send(:define_method, :to_s) do
      cs = champs.map { |c| "#{c}: #{send c}" }
      "<#{cs.join(', ')}>"
    end

    klass
  end
end
```

```ruby
# a.
Point = SClass.new(:x, :y, :z)
p Point.class, Point


# b.
p1 = Point[1, 0, 9]
p p1; puts p1



# c.
p [p1.z, p1.x]


# d.
p p1[:y]
```

5. *Pattern-matching* et `select` (5 pts)

Soit les deux classes ci-bas. Ces classes définissent d'autres méthodes, mais seules celles indiquées sont des méthodes que vous pourrez/devrez utiliser dans la question qui suit.

```ruby
class DateDeNaissance
  attr_reader :jour, :mois, :annee # => Integer
  ...
end

class Etudiant
  attr_reader :date_naissance # => DateDeNaissance

  def masculin? ...  # masculin? == !feminin?
  def feminin?  ...  # feminin?  == !masculin?
  ...
end
```

On veut définir une méthode `match_date_et_sexe` ayant l'interface suivante :

```ruby
# Retourne les étudiants dont la date de naissance et le sexe matchent
# ce qui est spécifié par les six premiers chiffres d'un code permanent
# style UQAM.
#
# @param [Array<Etudiant>] etudiants Une collection d'objets Etudiant
# @param [String] date_code_permanent Six premiers chiffres d'un code permanent
#
# @return [Array<Etudiant>]
#
def match_date_et_sexe( etudiants, date_code_permanent )
```

Des exemples d'appels sont présentés à la page suivante.

Complétez la mise en œuvre de la méthode `match_date_et_sexe` dont le squelette est présenté à la page suivante — les lignes en pointillés indiquent une partie à compléter, de même que les espaces indiqués par `# A COMPLETER`.

L'inteprétation d'un code permanent UQAM est comme suit, par ex., `DURN27510403` :

- `DUR` : Trois premières lettres du nom de famille, par ex., Durocher.
- `N` : Première lettre du prénom, par ex., Nellie.
- **275104** : Date de naissance sous forme `JJMMAA`. Dans le cas des femmes, on ajoute **50** au mois, femme née le 27 janvier 2004 = `275104`.
- `03` : Deux chiffres additionnels en cas de conflit (i.e., mêmes débuts de nom/prénom, même sexe et même date de naissance).

```
etudiants = [...] # Un Array d'objets Etudiant.

puts etudiants
#<Joe  Bidon (H): 06/12/1998>
#<Bill Smith (H): 15/06/1998>
#<John Doe   (H): 15/06/1998>
#<Jane Doe   (F): 15/06/1998>

puts match_date_et_sexe( etudiants, "150698" )
#<Bill Smith (H): 15/06/1998>
#<John Doe   (H): 15/06/1998>

puts match_date_et_sexe( etudiants, "155698" )
#<Jane Doe   (F): 15/06/1998>
```

```
# Motif pour décrire, de façon aussi précise que possible,
# les six premiers chiffres d'un code permanent.
DATE_DE_CP = ------------------------------------------------------------

def match_date_et_sexe( etudiants, date_code_permanent )
  m = DATE_DE_CP.match(date_code_permanent) # Objet MatchData

  fail "Date de code permanent invalide" ------------------------
  # A COMPLETER
```

```
  etudiants.select do |e|
    # A COMPLETER
```

```
  end
end
```

6. API coulante (DSL interne) pour décrire des cours (7 pts)

Le listing Ruby aux pages 16 et 17 présente une classe `Cours` comportant diverses méthodes. Ces méthodes peuvent être utilisées pour décrire des cours, et ce en utilisant une approche semblable à celle utilisée par `gli` pour décrire des commandes avec leurs options et actions associées — c'est-à-dire une approche avec constructeur et bloc, où une méthode sert à la fois de *getter* (si appel sans argument) et de *setter* (si appel avec argument).

---

[2] a) Donnez la mise en œuvre d'une méthode `Cours#prealables` (méthode d'instance de la classe `Cours`) qui permet d'obtenir les divers préalables d'un cours, tels que spécifiés par les appels faits avec la méthode `prealable` (sans `s`) :

```
# Définition de la méthode prealables
```

---

[5] b) Pour chaque série d'appels à la page 15, **indiquez (à droite) ce qui sera imprimé**.

Si une erreur est signalée, **indiquez quelle serait cette erreur** — en indiquant le message 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 16–17).

```ruby
# a.
inf1120 = Cours.creer do |c|
  c.sigle :INF1120
  c.titre "Prog. I"
end
puts inf1120



# b.
inf2120 = Cours.creer do |c|
  c.sigle :INF2120
  c.titre "Prog. II"
  c.prealables :INF1120
end
puts inf2120



# c.
inf3105 = Cours.creer do |c|
  c.sigle :INF3105
  c.prealable :INF1120
  c.prealable :INF2120
end
puts inf3105


# d.
inf600A = Cours.creer do |c|
  c.sigle :INF600A
  c.titre "Langages de script"
  c.prealable :INF1120
end
puts inf600A



# e.
puts Cours.avec(titre: "P[^I]*I")

Cours.avec(sigle: :INF1120) do |c|
  puts c.titre
end
```

```ruby
class Cours
  @les_cours = {}

  def initialize
    @prealables = []
  end
  private_class_method :new

  # Méthodes de classe.
  def self.creer
    nouveau_cours = new
    yield( nouveau_cours )
    unless nouveau_cours.valide?
      fail "*** #{self}.creer: non valide: #{nouveau_cours}"
    end
    @les_cours[nouveau_cours.sigle] = nouveau_cours

    nouveau_cours
  end

  def self.avec( sigle: nil, titre: nil )
    le_cours =
      if sigle
        @les_cours[sigle]
      else
        @les_cours.values.find { |c| /#{titre}/i =~ c.titre }
      end

    yield( le_cours ) if block_given?

    le_cours
  end
```

```ruby
  # Méthodes d'instance.
  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

  # Définition de la méthode prealables (définie plus haut).
  ...


  def to_s
    chaine_prealables = ''
    unless prealables.empty?
      chaine_prealables << ': ' << prealables.map(&:to_s).join(', ')
    end

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

  def valide?
    sigle && /^[A-Z]{3}[0-9]{4}$/ =~ sigle.to_s &&
      titre && !titre.empty? &&
      prealables.all? { |c| Cours.avec(sigle: c) }
  end
end
```