Écrire un premier programme
Calculer les facteurs d'un entier
Générer la suite de Fibonacci
Écrire une première classe
Créer une première hiérarchie de classes

Un premier programme

concepts :

fonction print

assignation

classe **dict**

énoncé while

function input

énoncé if, elif, else

moyenne.py

```python
# Un premier programme Python !
# François Major
# Pris dans Goodrich, Tamassia & Goldwasser
#   Data Structures & Algorithms in Python (c)2013
#   et modifié pour le système de notation de la FAS.

print( 'Bienvenue dans le calculateur moyenne générale !' )
print( 'Entrez vos notes littérales, une par ligne' )
print( 'et laissez la ligne blanche pour indiquer la fin.' )

# On utilise un dictionnaire pour convertir les lettres en points
points = { 'A+':4.3, 'A':4.0, 'A-':3.7, 'B+':3.3, 'B':3.0, 'B-':2.7,
           'C+':2.3, 'C':2.0, 'C-':1.7, 'D+':1.3, 'D':1.0, 'E':0.5, 'F':0.0 }

nombre_cours = 0
total_points = 0
termine = False

while not termine:
    note = input()
    if note == '':
        termine = True
    elif note not in points:
        #{0} accède le premier argument de format, ici note.
        print( "Note inconnue '{0}', sera ignorée".format( note ) )
    else:
        nombre_cours += 1
        total_points += points[ note ]

if nombre_cours > 0:
    print( 'Votre moyenne est {0:.3}'.format( total_points / nombre_cours ) )
    #{0:.3} est utilisé pour {argument:.format_spec}, soit l'argument passé à format
    #       et son format d'écriture, voir dans la documentation de Python
    #       "Format Specification Mini-Language"
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/moyenne.py
Bienvenue dans le calculateur moyenne générale !
Entrez vos notes littérales, une par ligne
et laissez la ligne blanche pour indiquer la fin.
B
C+
A
A-

Votre moyenne est 3.25
>>>
```

Facteurs d'un entier

concepts :
def
classe **list**
énoncé for
range
function append
énoncé return

facteurs_liste.py

```
"""Programme pour le cours IFT2015
   Écrit par François Major le 11 janvier 2014.

   Pris dans Goodrich, Tamassia & Goldwasser
     Data Structures & Algorithms in Python (c)2013

   Ce programme prend en input une valeur entière
   et retourne en output les facteurs de cette valeur.
"""

"""Fonction principale"""
def main():
    # Lire en entrée un entier
    print( "Ce programme retourne les facteurs d'un entier positif." )
    n = int( input( 'Entrez un entier positif: ' ) )

    """La fonction facteurs retourne les facteurs
       dans une liste.
    """
    print( 'Les facteurs de', n, 'sont :', facteurs( n ) )


"""Fonction facteurs retourne dans une liste
   les facteurs d'un entier positif n.
"""
def facteurs( n ):
    resultats = [] # liste initialisée vide
    for k in range( 1, n + 1 ):
        if n % k == 0:
            resultats.append( k )
    return resultats

"""Appeler la fonction principale"""
main()
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/facteurs_liste.py
Ce programme retourne les facteurs d'un entier positif.
Entrez un entier positif: 100
Les facteurs de 100 sont : [1, 2, 4, 5, 10, 20, 25, 50, 100]
>>>
```

Facteurs d'un entier

concepts :
function sort
generator
énoncé yield

facteurs_generateur.py

```
"""Programme pour le cours IFT2015
   Écrit par François Major le 11 janvier 2014.

   Pris dans Goodrich, Tamassia & Goldwasser
     Data Structures & Algorithms in Python (c)2013

   Ce programme prend en input une valeur entière
   et retourne en output les facteurs de cette valeur.
"""

"""Fonction principale"""
def main():
    # Lire en input un entier
    print( "Ce programme retourne les facteurs d'un entier positif." )
    n = int( input( 'Entrez un entier positif: ' ) )

    """La fonction facteurs retourne un générateur
       des facteurs de n.
       Pour former un output de type liste, on utilise
       un iterateur.
    """

    s = []
    for f in facteurs( n ):
        s.append( f )

    # on veut les sortir triés
    s.sort()
    print( s )


"""Fonction facteurs utilisant un générateur.
   Explore les valeurs jusqu'à la racine carré de n.
"""
def facteurs( n ):
    k = 1
    while k * k < n:
        if n % k == 0:
            yield k
            yield n // k
        k += 1
    if k * k  == n:
        yield k

"""Appeler la fonction principale"""
main()
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/facteurs_generateur.py
Ce programme retourne les facteurs d'un entier positif.
Entrez un entier positif: 100
[1, 2, 4, 5, 10, 20, 25, 50, 100]
>>>
```

Suite de Fibonacci

concepts :
énoncé break
infinite generator

fibonacci.py

```
"""Programme pour le cours IFT2015
   Écrit par François Major le 11 janvier 2014.

   Pris dans Goodrich, Tamassia & Goldwasser
     Data Structures & Algorithms in Python (c)2013

   Ce programme prend en input une valeur entière
   et retourne en output la suite de Fibonacci
   jusqu'à cette valeur.
"""

"""Fonction principale"""
def main():
    # Lire en input un entier
    n = int( input( 'Entrez un entier positif: ' ) )

    """La fonction fibonacci retourne un générateur
       des nombres de la suite jusqu'à l'infini.
    """

    s = "["
    for fibo in fibonacci():
        if fibo > n:
            break
        s += str( fibo ) + ", "
    s += "... ]"
    print( s )


"""Fonction fibonacci utilisant un generateur.
"""
def fibonacci( ):
    a = 0
    b = 1
    while True:
        yield a
        portee = a + b # Fibonacci calcule portée de lapins !
        a = b
        b = portee

"""Appeler la fonction principale"""
main()
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/fibonacci.py
Entrez un entier positif: 100
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... ]
>>>
```

kième élément de la suite de Fibonacci

concepts :
entiers très grands

fibonaccik.py

```
"""Programme pour le cours IFT2015
   Écrit par François Major le 11 janvier 2014.

   Pris dans Goodrich, Tamassia & Goldwasser
     Data Structures & Algorithms in Python (c)2013

   Ce programme prend en input une valeur entière, k,
   et retourne en output le kième terme de la suite
   de Fibonacci
"""

"""Fonction principale"""
def main():

    """Accéder le kè élément de la suite de Fibonacci"""
    # Lire en input un entier
    print( "Accède au kè élément de la suite de Fibonacci." )
    n = int( input( 'Entrez un entier positif k : ' ) )

    k = 1
    for fibo in fibonacci():
        if k == n:
            s = "[" + str( fibo ) + "]"
            break
        k += 1
    print( s )

"""Fonction fibonacci utilisant un generateur.
"""
def fibonacci( ):
    a = 0
    b = 1
    while True:
        yield a
        portee = a + b # Fibonacci calcule portée de lapins !
        a = b
        b = portee

"""Appeler la fonction principale"""
main()
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/fibonaccik.py
Accède au kè élément de la suite de Fibonacci.
Entrez un entier positif k : 100
[218922995834555169026]
>>>
```

Nombres pairs de la suite de Fibonacci

concepts :
syntaxe de compréhension de liste

fibopairs.py

```
"""Programme pour le cours IFT2015
   Écrit par François Major le 15 janvier 2014.

   Pris dans Goodrich, Tamassia & Goldwasser
     Data Structures & Algorithms in Python (c)2013

   Ce programme prend en input une valeur entière
   et retourne les nombres paires de la suite de
   Fibonacci en output jusqu'à cette valeur.
"""

"""Fonction principale"""
def main():
    # Lire en input un entier
    n = int( input( 'Entrez un entier positif: ' ) )

    """La fonction fibonacci retourne un générateur
       des nombres de la suite jusqu'à l'infini.
    """

    s = "["
    for fibo in (f for f in fibonacci() if f % 2 == 0):
        if fibo > n:
            break
        s += str( fibo ) + ", "
    s += "... ]"
    print( s )

"""Fonction fibonacci utilisant un generateur.
"""
def fibonacci( ):
     a = 0
     b = 1
     while True:
         yield a
         portee = a + b # Fibonacci calcule portée de lapins !
         a = b
         b = portee

"""Appeler la fonction principale"""
main()
```

```
"""La syntaxe de compréhension permet de produire
   4 types de contenant:

   [ k*k for k in range( 1, n+1 ) ] - liste
   { k*k for k in range( 1, n+1 ) } - ensemble
   ( k*k for k in range( 1, n+1 ) ) - générateur
   { k : k*k for k in range( 1, n+1 ) } - dictionnaire

   Dans le cas où on parcourt un générateur infini,
   on doit utiliser le générateur car si on utilise
   les 3 autres, alors python va vouloir compléter
   la liste, l'ensemble ou le dictionnaire pour la
   suite infinie.
"""
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/fibopairs.py
Entrez un entier positif: 100
[0, 2, 8, 34, ... ]
>>>
```

classe Intervalle

concepts :

constructor

iterator

unit testing

Intervalle.py

```
"""Classe pour le cours IFT2015
   Écrit par Francois Major le 12 janvier 2014.

   Cette classe définit un intervalle, soit
   l'équivalent du "range" en python.

   Pris dans Goodrich, Tamassia & Goldwasser
        Data Structures & Algorithms in Python (c)2013

   Comportement :
     Intervalle( sup )           = 0, 1, ... sup-1
     Intervalle( inf, sup )      = inf, inf+1, ... sup-1
     Intervalle( inf, sup, inc ) = inf, inf+inc, inf+2*inc, ... sup-1
"""
```

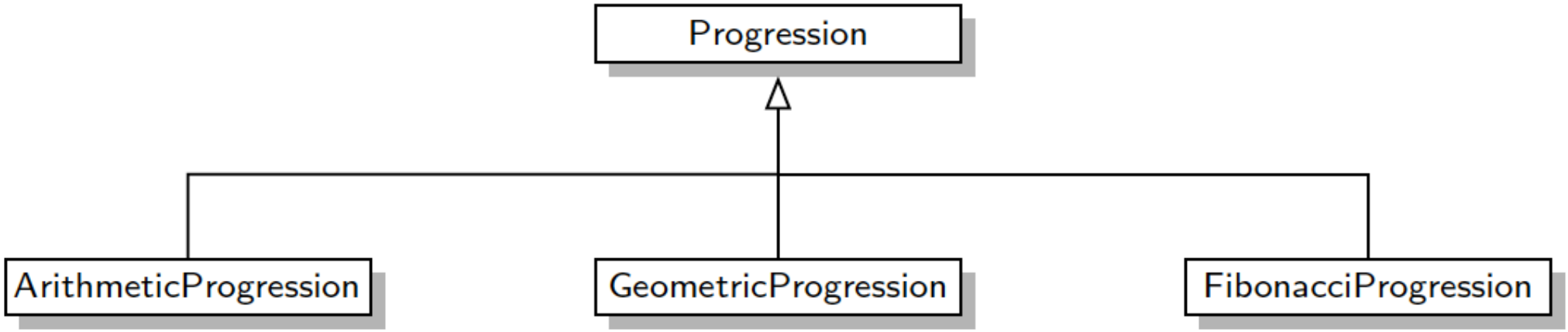
```python
class Intervalle:

    """Un intervalle se définit par:
       - une valeur inf
       - une valeur sup
       - une valeur inc (increment)
    """

    """Constructeur
    """
    def __init__( self, inf, sup = None, inc = 1 ):

        if inc == 0:
            raise ValueError( "L'incrément ne peut pas être 0" )

        if sup is None: #cas range( sup )
            inf, sup = 0, inf

        self._length = max( 0, ( sup - inf + inc - 1 ) // inc )

        self._inf = inf
        self._sup = sup
        self._inc = inc

    """Les methodes
    """

    """__str__ retourne une chaîne lisible
    """
    def __str__( self ):
        return "intervalle( " + str( self._inf ) + ", " + str( self._sup ) + ", " + str( self._inc ) + " )"

    """Une classe qui définit __len__ et __getitem__ fournit automatiquement un itérateur

       __len__ retourne le nombre d'éléments dans l'intervalle
       __getitem__ retourne le kème élément de l'intervalle
    """

    def __len__( self ):
        return self._length

    def __getitem__( self, k ):
        # si k < 0, index à partir de la fin de l'intervalle
        if k < 0:
            k += len( self )

        # si k indice un élément en dehors de l'intervalle,
        # alors il faut lancer une exception.
        if not 0 <= k < self._length:
            raise IndexError( 'index hors limite' )

        # calcule et retourne le kè élément
        return self._inf + k * self._inc

"""Fin class intervalle:
"""
```

```
"""Validation de la classe (unit testing)
"""

if __name__ == '__main__':

    int1 = Intervalle( 10 )
    print( int1 )

    int2 = Intervalle( 0, 10 )
    print( int2 )

    int3 = Intervalle( 0, 10, 2 )
    print( int3 )

    for i in int3:
        print( i )

    int4 = Intervalle( -10, 10, 2 )
    print( int4 )

    for i in int4:
        print( i )

"""Fin Validation de la classe (unit testing)
"""
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/Intervalle.py
intervalle( 0, 10, 1 )
intervalle( 0, 10, 1 )
intervalle( 0, 10, 2 )
0
2
4
6
8
intervalle( -10, 10, 2 )
-10
-8
-6
-4
-2
0
2
4
6
8
>>>
```

Progressions

concept :
héritage



Progression.py

```python
# Classe Python pour des progressions numériques
# François Major
# Pris dans Goodrich, Tamassia & Goldwasser
#   Data Structures & Algorithms in Python (c)2013

class Progression:
    """Un itérateur produisant une progression numérique
       L'itérateur par défaut produit les nombres 0, 1, 2, ...
    """

    def __init__( self, debut = 0 ):
        #Initialise courant à la première valeur de la progression
        self._courant = debut

    def _avance( self ):
        """Met à jour self._courant à une nouvelle valeur

        Cette méthode sera redéfinie dans la sous-classe.

        Par convention, si courant est mis à None, cela signifie
        la fin d'une progression finie.
        """

        self._courant += 1

    def __next__( self ):
        #Retourne le prochain élément, ou soulève une error StopIteration.
        if self._courant is None:   #convention pour terminer la progression
            raise StopIteration()
        else:
            reponse = self._courant #enregistre la valeur courante à retourner
            self._avance()          #avance pour préparer la prochaine valeur
            return reponse          #retourne la reponse

    def __iter__( self ):
        #Par convention, un itérateur doit retourner lui-même comme itérateur.
        return self

    def print_progression( self, n ):
        #Imprime les prochaines n valeurs de la progression.
        print( ' '.join( str( next( self ) ) for j in range( n ) ) )


if __name__ == '__main__':
    print( "Progressions par défaut :" )
    Progression().print_progression( 10 )

    print( "Progressions 11 :" )
    Progression( 11 ).print_progression( 10 )
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/Progression.py
Progressions par défaut :
0 1 2 3 4 5 6 7 8 9
Progressions 11 :
11 12 13 14 15 16 17 18 19 20
>>>
```

Progressions arithmétiques

concepts :
from… import…
overriding

Progression_Arithmetique.py

```python
from Progression import Progression

# Classe Python pour des progressions arithmétiques
# François Major
# Pris dans Goodrich, Tamassia & Goldwasser
#   Data Structures & Algorithms in Python (c)2013

class Progression_Arithmetique( Progression ): #hérite de la classe Progression
    #Itérateur pour produire une progression arithmétique

    def __init__( self, increment = 1, debut = 0 ):
        """Crée une nouvelle progression arithmétique

        increment   par une constante fixe ajoutée à chaque terme (défaut 1)
        debut       le premier terme de la progression (défaut 0)
        """

        super().__init__( debut )
        self._increment = increment

    def _avance( self ):
        #Met à jour la valeur courante en ajoutant l'incrément fixe.
        self._courant += self._increment

if __name__ == '__main__':
    print( "Progression arithmétique avec incrément de 5 :" )
    Progression_Arithmetique( 5 ).print_progression( 10 )

    print( "Progression arithmétique avec incrément de 5 débutant à 2 :" )
    Progression_Arithmetique( 5, 2 ).print_progression( 10 )
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/Progression_Arithmetique.py
Progression arithmétique avec incrément de 5 :
0 5 10 15 20 25 30 35 40 45
Progression arithmétique avec incrément de 5 débutant à 2 :
2 7 12 17 22 27 32 37 42 47
>>>
```

```python
from Progression import Progression

# Classe Python pour des progressions géométriques
# François Major
# Pris dans Goodrich, Tamassia & Goldwasser
#   Data Structures & Algorithms in Python (c)2013

class Progression_Geometrique( Progression ): #hérite de la classe Progression
    #Itérateur pour produire une progression géométrique

    def __init__( self, base = 2, debut = 1 ):
        """Crée une nouvelle progression géométrique

        base        une constante fixe multipliée à chaque terme (défaut 2)
        debut       le premier terme de la progression (défaut 1)
        """

        super().__init__( debut )
        self._base = base

    def _avance( self ):
        #Met à jour la valeur courante en multipliant par la base fixe.
        self._courant *= self._base

if __name__ == '__main__':
    print( "Progression géométrique par défaut :" )
    Progression_Geometrique().print_progression( 10 )

    print( "Progression arithmétique avec base de 3 :" )
    Progression_Geometrique( 3 ).print_progression( 10 )
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/Progression_Geometrique.py
Progression géométrique par défaut :
1 2 4 8 16 32 64 128 256 512
Progression arithmétique avec base de 3 :
1 3 9 27 81 243 729 2187 6561 19683
>>>
```

```python
from Progression import Progression

# Classe Python pour des progressions de type Fibonacci
# François Major
# Pris dans Goodrich, Tamassia & Goldwasser
#   Data Structures & Algorithms in Python (c)2013

class Progression_Fibonacci( Progression ): #hérite de la classe Progression
    #Itérateur pour produire une progression Fibonacci

    def __init__( self, premier = 0, deuxieme = 1 ):
        """Crée une nouvelle progression Fibonacci

        premier         premier terme de la progression (défaut 0)
        deuxieme        deuxième terme de la progression (défaut 1)
        """

        super().__init__( premier )          #démarre la progression au premier terme
        self._precedent = deuxieme - premier #valeur précédente de la première

    def _avance( self ):
        #Met à jour la valeur courante en additionnant les 2 valeurs précédentes
        self._precedent, self._courant = self._courant, self._precedent + self._courant

if __name__ == '__main__':
    print( "Progression Fibonacci par défaut :" )
    Progression_Fibonacci().print_progression( 10 )

    print( "Progression Fibonacci en partant avec 4 et 6 :" )
    Progression_Fibonacci( 4, 6 ).print_progression( 10 )
```

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/major/Dropbox/Documents/Cours/IFT2015/H18/Éléments fondamentaux
/Progression_Fibonacci.py
Progression Fibonacci par défaut :
0 1 1 2 3 5 8 13 21 34
Progression Fibonacci en partant avec 4 et 6 :
4 6 10 16 26 42 68 110 178 288
>>>
```