TP : Héritage, liaison dynamique.

POO : Programmation Orientée Objet

1A, 2A ou 3A

L'objectif de ce TP est de concevoir un programme permettant de modéliser les dipôles électriques, puis de calculer leur *impédance*.

1 Le problème

Les dipôles sont composés de composants "élémentaires" : *résistance*, *self* et *capacité*. Ces composants élémentaires peuvent être combinés par un montage *en série* ou *en parallèle*.

Soit ω un nombre réel appelé *pulsation* et i le nombre complexe de module 1 et d'argument $\pi/2$. *L'impédance* z d'un dipôle est un nombre complexe.

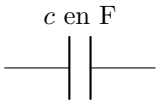
Une *résistance* de valeur r exprimée en ohms (symbole Ω)

$z = r$

Une *self* de valeur l exprimée en henrys (symbole H)

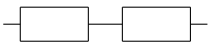
$z = i(\omega * l)$

Une *capacité* de valeur c exprimée en farad (symbole F)



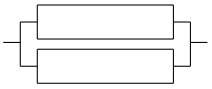
$z = i(\frac{-1}{\omega * c})$

Un dipôle composé de deux dipôles d'impédance z_1 et z_2 montés *en série*



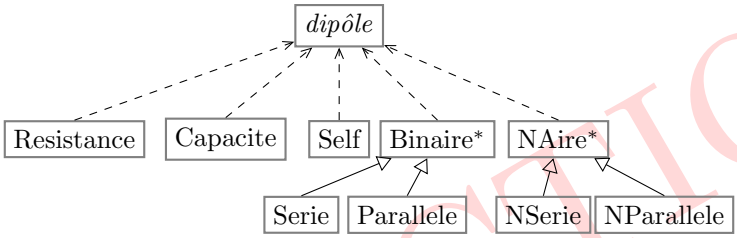
$z = z_1 + z_2$

Un dipôle composé de deux dipôles d'impédance z_1 et z_2 montés *en parallèle*



$z = \frac{1}{\frac{1}{z_1} + \frac{1}{z_2}}$

Nous pouvons modéliser les dipôles électriques en utilisant la hiérarchie de classes suivante :



Chaque classe définira la méthode `impedance(omega)` permettant d'évaluer l'impédance du dipôle manipulé (le paramètre `omega` représentant la pulsation sous la forme d'un nombre réel).

L'impédance d'un dipôle est en général un nombre complexe. On peut remarquer que l'impédance d'une résistance est un nombre réel et que l'impédance d'une self ou d'une capacité sont des nombres complexes imaginaires purs.

2 Sujet du TP

Élements fournis. La classe `Complexe` qui modélise un nombre complexe. Cette classe permet de faire des calculs de base sur les nombres complexes.

Réponse

```
from math import sqrt, atan2

class Complexe:

    def __init__(self, re, im):
        # constructeur
        self.re = float(re) # partie reelle
        self.im = float(im) # partie imaginaire
```

```
    def __str__(self):
        # sous forme de chaine de caractere
        return "%g + %gi" % (self.re,self.im)

    def add(self, other):
        return Complexe(self.re + other.re, self.im + other.im)

    def sub(self, other):
        return Complexe( self.re - other.re, self.im - other.im)

    def mul(self, other):
        return Complexe(self.re*other.re - self.im*other.im,  self.re*other.im + self.im*other.re)

    def div(self, other):
        denominator = other.re ** 2 + other.im ** 2
        return Complexe( \
            (self.re * other.re + self.im * other.im) / denominator,
            (self.im * other.re - self.re * other.im) / denominator )

    def conj(self):
        return Complexe(self.re, -self.im)

    def abs(self):
        return sqrt(self.re ** 2 + self.im ** 2)

    def arg(self):
        return atan2(self.im, self.re)

    def inv(self):
            m = self.re ** 2 + self.im ** 2
            return Complexe(self.re / m, - self.im / m)

    def __eq__(self, other):
        return self.__dict__ == other.__dict__
        #return self.im == other.im and self.re == other.re

    def __ne__(self, other):
            return not self == other

if __name__ == "__main__" :
    from math import pi

    x = Complexe(0, 1)
    y = Complexe(1, 0)
    z = Complexe(1, 1)

    print "x =", x
    print "y =", y
    print "z =", z
    print

    print "x + y =", z
    print "x - y =", x.sub(y)
    print "x * y =", x.mul(y)
    print "x * z =", x.mul(z)
    print "x / x =", x.div(x)
    print "x / z =", x.div(z)
    print "conj(z) =", z.conj()
    print "abs(z)**2 =", z.abs()**2
    print "check:", z.mul(z.conj())
    print "arg(z) =", z.arg() * 180/pi
    print "inv(x) = ", x.inv()
    print "inv(y) = ", y.inv()
    print "inv(z) = ", z.inv()
```

Fin réponse

Réflexion préliminaire à mener. Il est important qu'avant de commencer à programmer quoique ce soit, vous parcouriez le code source des classes fournies pour avoir un aperçu des méthodes que vous pouvez utiliser et celles que vous devrez implanter.

Notation. Dans le langage Python, la valeur 10e-2 peut s'écrire `1e-2`

★ **Exercice 1.** Nous souhaitons tout d'abord manipuler des dipôles élémentaires.

▷ **Question 1.** Etudier la classe `Resistance` implantant le dipôle élémentaire résistance ainsi que sa méthode pour calculer son impédance.

Réponse

```
from Complexe import *

class Resistance:

    def __init__(self, v):
        # constructeur
        self.value = float(v)

    def __str__(self):
        # sous forme de chaine de caractere
        return "Resistance(%g)" % (self.value)

    def impedance(self, omega):
        return Complexe(self.value, 0)

if __name__ == "__main__" :
        r = Resistance(1e2)
        expected_result = Complexe(100, 0)
        result = r.impedance(314.16)
        print "got " + str(result) + ", expected " + str(expected_result)


```

Fin réponse

▷ **Question 2.** En s'inspirant de la classe `Resistance` compléter le code de la classe `Self` permettant de calculer son impédance.

▷ **Question 3.** Vérifiez le bon fonctionnement de votre code grâce au test qui calcule l'impédance du dipôle figure **??** pour $\omega = 314.16$. Le résultat escompté est environ $0.0 + 21.99i$.

Réponse

Self.java

```
from Complexe import *

class Self:

    def __init__(self, v):
        # constructeur
        self.value = float(v)

    def __str__(self):
        # sous forme de chaine de caractere
        return "Self(%g)" % (self.value)

    def impedance(self, omega):
        return Complexe(0, self.value*omega)

if __name__ == "__main__" :
        r = Self(7e-2)
        expected_result = Complexe(0, 21.9912)
        result = r.impedance(314.16)
        print "got " + str(result) + ", expected " + str(expected_result)


```

Fin réponse

▷ **Question 4.** De la même façon, écrivez la classe `Capacite` implantant le dipôle élémentaire capacité ainsi que sa méthode pour calculer son impédance.

Réponse

```
from Complexe import *

class Capacite:

    def __init__(self, v):
        # constructeur
        self.value = float(v)

    def __str__(self):
        # sous forme de chaine de caractere
```

```
        return "Capacite(%g)" % (self.value)

    def impedance(self, omega):
        return Complexe(0, -1 / (self.value*omega))

if __name__ == "__main__" :
        r = Capacite(42)
        expected_result = Complexe(0, 7.578789e-5)
        result = r.impedance(314.16)
        print "got " + str(result) + ", expected " + str(expected_result)


```

Fin réponse

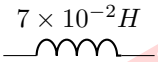


FIGURE 1 – La self testée.

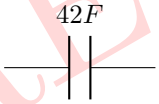


FIGURE 2 – La capacité testée.

FIGURE 3 – La résistance testée.

★ **Exercice 2.** Nous souhaitons maintenant construire des dipôles combinant plusieurs dipôles (élémentaires ou résultant déjà d'une composition).

▷ **Question 5.** Écrivez la classe `Serie` permettant de définir un dipôle composé de deux dipôles en série et son impédance.

Réponse

```
from Complexe import *

class Serie:

    def __init__(self, dipole1, dipole2):
        # constructeur
        self.d1 = dipole1
        self.d2 = dipole2

    def __str__(self):
        # sous forme de chaine de caractere
        return "Serie(" + self.d1 + ", " + self.d2 + ")"

    def impedance(self, omega):
        return self.d1.impedance(omega).add(self.d2.impedance(omega))

if __name__ == "__main__" :
        from Self import *
        from Resistance import *

        r = Serie(Self(5e-2), Resistance(1e2))
        expected_result = Complexe(100, 15.708)
        result = r.impedance(314.16)
        print "got " + str(result) + ", expected " + str(expected_result)


```

Fin réponse

▷ **Question 6.** Vérifiez le bon fonctionnement de votre code grâce au calcul de l'impédance du dipôle de la figure **??** pour $\omega = 314.16$ *(résultat escompté* $\approx$ *100.0 + 15.70i).*

▷ **Question 7.** De la même façon, écrivez la classe `Parallele` implantant un dipôle combinant deux dipôles montés en parallèle et permettant de calculer son impédance.

Réponse

```
from Complexe import *

class Parallele:

```

```
    def __init__(self, dipole1, dipole2):
        # constructeur
        self.d1 = dipole1
        self.d2 = dipole2

    def __str__(self):
        # sous forme de chaine de caractere
        return "Parallel(" + self.d1 + ", " + self.d2 + ")"

    def impedance(self, omega):
        c1 =  self.d1.impedance(omega)
        c2 =  self.d2.impedance(omega)
        return c1.inv().add(c2.inv()).inv()

if __name__ == "__main__" :
        from Self import *
        from Capacite import *
        from Resistance import *

        r = Parallele(Self(5e-2), Capacite(9e-4))
        expected_result = Complexe(0, -4.564497387210561)
        result = r.impedance(314.16)
        print "got " + str(result) + ", expected " + str(expected_result)

        r = Parallele(Resistance(1e2), Parallele(Self(5e-2), Capacite(9e-4)))
        expected_result = Complexe(0.2079131844185524, -4.55500719534011)
        result = r.impedance(314.16)
        print "got " + str(result) + ", expected " + str(expected_result)


```

Fin réponse

▷ **Question 8.** Vérifiez le bon fonctionnement de votre code grâce au calcul de l'impédance du dipôle de la figure **??** pour $\omega = 314.16$ *(résultat escompté ≈ 0.2079 + -4.55i).*

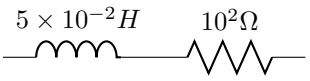


FIGURE 4 – Montage en série testé.

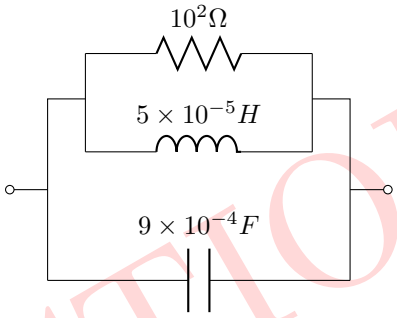


FIGURE 5 – Montage en parallèle testé.

★ **Exercice 3.** On voudrait maintenant pouvoir modéliser la mise en série ou en parallèle d'un nombre quelconque de dipôles.

▷ **Question 9.** Écrivez la classe `NSerie` pour définir un dipôle composé d'un nombre quelconque de dipôles en série et le calcul de son impédance. L'impédance de n dipôles d'impédance ω_i montés en série peut se calculer en utilisant la formule suivante : $\sum_{i=1}^{n} \omega_i$.

Réponse

TODO ;)

Fin réponse

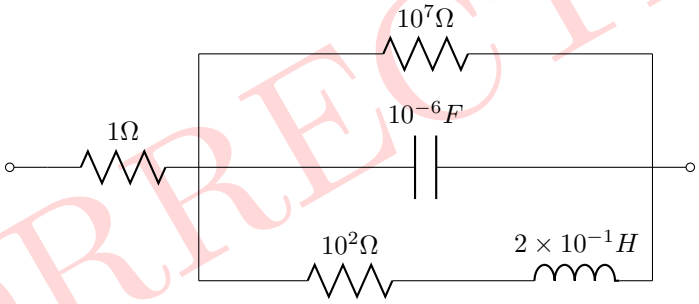
▷ **Question 10.** Écrivez la classe `NParallele` pour définir un dipôle composé d'un nombre quelconque de dipôles en parallèle et le calcul de son impédance. L'impédance de n dipôles d'impédance ω_i montés en parallèle peut se calculer en utilisant la formule suivante : $\frac{1}{\sum_{i=1}^{n} \frac{1}{\omega_i}}$.

Réponse

TODO ;)

Fin réponse

▷ **Question 11.** Testez votre code grâce au dipôle suivant en utilisant ces nouveaux types (sans utiliser les types `Serie` ou `Parallele` précédemment définis) et de calculer son impédance pour $\omega = 314.16$. *Résultat escompté* $\approx 104.9604 + 60.764i$.



★ **Exercice 4.** L'objectif est maintenant de vous faire *instancier* des dipôles complexes.

▷ **Question 12.** Ecrire un programme permettant d'instancier les dipôles des figures **??** et **??**.

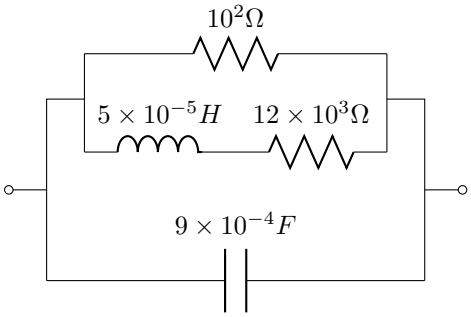


FIGURE 6 – Le dipôle `dip1` à réaliser.

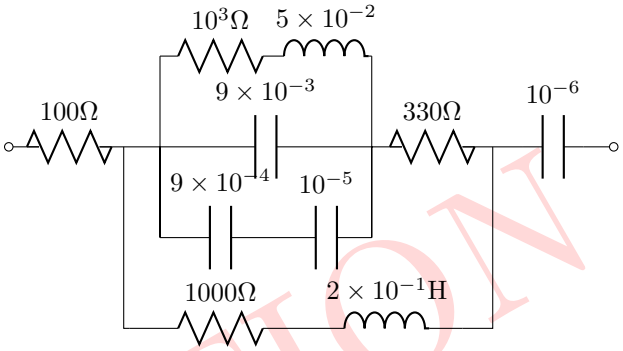


FIGURE 7 – Le dipôle `dip2` à réaliser.