Herramientas Computacionales
2016661
Introducción a la programación orientada a objetos

Ricardo Amézquita

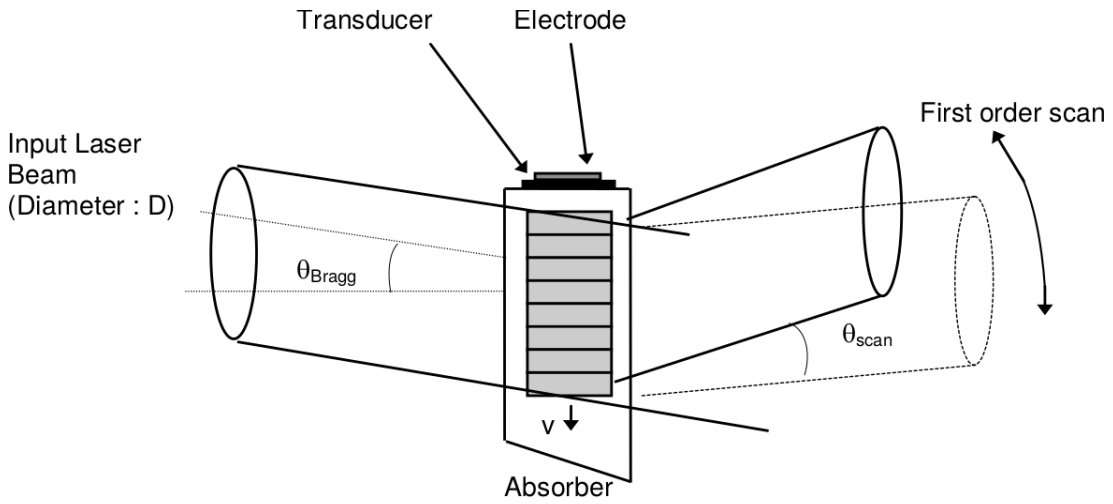
Departamento de Física
Universidad Nacional de Colombia
Sede Bogotá

1. Primera aproximación a la POO

2. Operaciones entre números complejos - Ejemplo 2
   - Ejemplo usando funciones
   - Primer ejemplo con objetos

Primera aproximación a la POO

Calculo de los parámetros de operación de un AOD



Tomado de:http://www.isomet.com/App-Manual_pdf/AO%20Modulation.pdf

Parámetros y Ecuaciones para un AOD

Access Time T_a	15 μs
Modulation Frecuency F_m	140 MHz
BandWidth ΔF	40 MHz
Acoustic Velocity V_a	650 m/s
Illumination Wavelenght λ	405 nm
Tiempo de escaneo T_{scan}	0.01 ~ 1 ms

Resolución Máxima	$TBP = T_a \Delta F$
Resolución Real	$N_{spots} = TBP \left[1 - \frac{T_a}{T_{Scan} - T_a}\right]$
Ángulo de Bragg	$\Theta_{Bragg} = \frac{\lambda F_m}{2 V_a}$
Ángulo de escaneo	$\Theta_{Scan} = \lambda \frac{\Delta F}{V_a}$
Distancia escaneo	$Max_{Scan} = \Theta_{Scan} F_{obj}$
Resolución Máxima	$Res_{Max} = F_{obj} \Theta_{Scan} / N_{spots}$

Problema 1:

Hacer una gráfica del numero total de puntos en función de la frecuencia de escaneo, y a partir de esta encontrar la frecuencia de escaneo para la cual se maximiza la velocidad de impresión en puntos por segundo.
Nota: una vez tengan el programa que genera la tabla de datos, preguntar como se puede hacer la gráfica.

Problema 2:

Suponga ahora que usted tiene 4 moduladores diferentes con los siguientes parámetros:

Access Time T_a	15 μs	25 μs	15 μs	20 μs
Modulation Frecuency F_m	140 MHz	140 MHz	160 MHz	160 MHz
BandWidth ΔF	40 MHz	40 MHz	60 MHz	60 MHz
Acoustic Velocity V_a	650 m/s	650 m/s	650 m/s	650 m/s
Illumination Wavelenght λ	405 nm	405 nm	405 nm	405 nm
Tiempo de escaneo T_{scan}	0.01 ~ 1 ms	0.01 ~ 1 ms	0.01 ~ 1 ms	0.01 ~ 1 ms

Haga un programa (un solo programa, no 4 programas) que repita la gráfica encontrada en el punto 1, para los 4 deflectores.

Definición

La programación orientada a objetos o POO (OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento. Su uso se popularizó a principios de la década de los años 1990. En la actualidad, existe variedad de lenguajes de programación que soportan la orientación a objetos.

Tomado de: http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos

Operaciones entre números complejos - Ejemplo 2

Ejemplo de programa sin objetos

ComplexFunc.py parte 1

```
11 def suma(a,b):
12     """
13     Funci_n que calcula la suma de 2 n_meros complejos
14     """
15     are=a[0]
16     aim=a[1]
17     bre=b[0]
18     bim=b[1]
19     return (are+bre,aim+bim)

21 def multip(a,b):
22     """
23     Funci_n que calcula el producto de 2 n_meros complejos
24     """
25     are=a[0]
26     aim=a[1]
27     bre=b[0]
28     bim=b[1]
29     return (are*bre-aim*bim,aim*bre+bim*are)
```

Ejemplo de programa sin objetos

ComplexFunc.py parte 2

```
31 def divi(a,b):
32     """Funci_n que calcula la divici_n entre 2 numeros complejos
33     a/b"""
34     are=a[0]
35     aim=a[1]
36     bre=b[0]
37     bim=b[1]
38     return (are*bre+aim*bim)/(bre**2+bim**2),
39            (aim*bre-are*bim)/(bre**2+bim**2)

41 def imp(a):
42     """Funci_n que genera una cadena de caracteres con la
43     representaci_n de un n_mero complejo
44     """
45     return "{:f}{:+f}i".format(a[0],a[1])
```

Ejemplo de programa sin objetos

ComplexFunc.py parte 3

```
47 if __name__=="__main__":
48     q=(1.,5.)
49     w=(3.,4.)
50     h=(4.,0)

52     a=suma(q,w)
53     b=multip(q,h)
54     c=divi(w,h)

56     print imp(a)
57     print imp(b)
58     print imp(c)
```

Primera aproximación a un programa con objetos

ComplexClassIntro.py parte 1

```
8  class Complex:
9      def __init__(self,re,im=0):
10         """Define un numero complejo, donde re contiene la parte
11         real e im contiene la parte imaginaria del n_mero.
12         """
13         self.re=re
14         self.im=im

16     def suma(self,other):
17         """
18         Funci_n que calcula la suma de 2 n_meros complejos
19         """
20         return Complex(self.re+other.re,self.im+other.im)

22     def multip(self,other):
23         """
24         Funci_n que calcula el producto de 2 n_meros complejos
25         """
26         return Complex(self.re*other.re-self.im*other.im,
```

Primera aproximación a un programa con objetos

ComplexClassIntro.py parte 2

```
29      def divi(self,other):
30          """Funci_n que calcula la divici_n entre 2 numeros
31          complejos self/other"""
32          return Complex((self.re*other.re+self.im*other.im)/\
33                         (other.re**2+other.im**2),
34                         (self.im*other.re-self.re*other.im)/\
35                         (other.re**2+other.im**2))

37      def imp(self):
38          """Funci_n que genera una cadena de caracteres con la repre
39          de un n_mero complejo
40          """
41          return "{:f}{:+f}i".format(self.re,self.im)
```

Primera aproximación a un programa con objetos

ComplexClassIntro.py parte 3

```
43 if __name__=="__main__":
44     q=Complex(1.,5.)
45     w=Complex(3.,4.)
46     h=Complex(4.)

48     a=q.suma(w)
49     b=q.multip(h)
50     c=w.divi(h)

52     print a.imp()
53     print b.imp()
54     print c.imp()
```

Problema

Con alguna de las 2 librerías de calculo aritmético entre números complejos realice las siguiente operación:

$$-2 \times (32 + 43\imath) \times (45 + 3\imath) \,/\, ((24 - 5\imath) \times (100 + 56\imath)) + 5$$

¿Que dificultades encontró al escribir la expresión necesaria para realizar el calculo?

Sobrecarga de operadores

ComplexClassOverloading.py parte 1

```
12 class Complex:
13     def __init__(self,re,im=0):
14         """Define un numero complejo, donde re contiene la parte
15         real e im contiene la parte imaginaria del n_mero.
16         """
17         self.re=re
18         self.im=im

20     def __add__(self,other):
21         """
22         Funci_n que calcula la suma de 2 n_meros complejos
23         """
24         return Complex(self.re+other.re,self.im+other.im)

26     def __sub__(self,other):
27         """
28         Funci_n que calcula la suma de 2 n_meros complejos
29         """
30         return Complex(self.re-other.re,self.im-other.im)
```

Sobrecarga de operadores

ComplexClassOverloading.py parte 2

```
32      def __neg__(self):
33          return Complex(-self.re,-self.im)

35      def __mul__(self,other):
36          """
37          Funci_n que calcula el producto de 2 n_meros complejos
38          """
39          return Complex(self.re*other.re-self.im*other.im,
40                         self.im*other.re+other.im*self.re)

42      def __div__(self,other):
43          """Funci_n que calcula la divici_n entre 2 numeros
44          complejos self/other"""
45          return Complex((self.re*other.re+self.im*other.im)/\
46                         (other.re**2+other.im**2),
47                         (self.im*other.re-self.re*other.im)/\
48                         (other.re**2+other.im**2))
```

Sobrecarga de operadores

ComplexClassOverloading.py parte 3

```
50      def __str__(self):
51          """Funci_n que genera una cadena de caracteres con
52          la representaci_n de un n_mero complejo
53          """
54          return "{:f}{:+f}i".format(self.re,self.im)

56  if __name__=="__main__":
57      q=Complex(1.,5.)
58      w=Complex(3.,4.)
59      h=Complex(4.)

61      a=q+w
62      a1=q-w
63      a3=-w
64      b=q*h
65      c=w/h

67      print a
68      print a1
69      print b
70      print c
```

Problema

Usando la librería de calculo aritmético entre números complejos que usa operadores sobrecargados, repita la operación:

$$-2 \times (32 + 43\imath) \times (45 + 3\imath) \,/\, ((24 - 5\imath) \times (100 + 56\imath)) + 5$$

¿Que dificultades encontró al escribir la expresión necesaria para realizar el calculo?

Una mejor sobrecarga de operadores

ComplexClassBetterOverloading.py parte 1

```
12 class Complex:
13     def __init__(self,re,im=0):
14         """Define un numero complejo, donde re contiene la parte
15         real e im contiene la parte imaginaria del n_mero.
16         """
17         self.re=re
18         self.im=im
```

Una mejor sobrecarga de operadores

ComplexClassBetterOverloading.py parte 2

```
20      def __add__(self,other):
21          """
22          Funci_n que calcula la suma de 2 n_meros complejos
23          """
24          if isinstance(other,Complex):
25              return Complex(self.re+other.re,self.im+other.im)
26          elif isinstance(other,(int,float)):
27              return Complex(self.re+other,self.im)
28          else: return NotImplemented

30      def __radd__(self,other):
31          """
32          Funci_n que calcula la suma de 2 n_meros complejos
33          """
34          return self.__add__(other)
```

Una mejor sobrecarga de operadores

ComplexClassBetterOverloading.py parte 3

```
36      def __sub__(self,other):
37          """
38          Funci_n que calcula la suma de 2 n_meros complejos
39          """
40          if isinstance(other,Complex):
41              return Complex(self.re-other.re,self.im-other.im)
42          elif isinstance(other,(int,float)):
43              return Complex(self.re-other.re,self.im)
44          else: return NotImplemented

46      def __rsub__(self,other):
47          return self.__sub__(other)


50      def __neg__(self):
51          return Complex(-self.re,-self.im)
```

Una mejor sobrecarga de operadores

ComplexClassBetterOverloading.py parte 4

```
53      def __mul__(self,other):
54          """
55          Funci_n que calcula el producto de 2 n_meros complejos
56          """
57          if isinstance(other,Complex):
58              return Complex(self.re*other.re-self.im*other.im,
59                             self.im*other.re+other.im*self.re)
60          elif isinstance(other,(int,float)):
61              return Complex(other*self.re,other*self.im)
62          else: return NotImplemented

64      def __rmul__(self, other):
65          return self.__mul__(other)
```

Una mejor sobrecarga de operadores

ComplexClassBetterOverloading.py parte 5

```
67      def __div__(self,other):
68          """Funci_n que calcula la divici_n entre 2 numeros
69          complejos self/other"""
70          if isinstance(other,Complex):
71              return Complex((self.re*other.re+self.im*other.im)/\
72                             (other.re**2+other.im**2),
73                             (self.im*other.re-self.re*other.im)/\
74                             (other.re**2+other.im**2))
75          elif isinstance(other,(int,float)):
76              return Complex(self.re/other, self.im/other)
77          else: return NotImplemented

79      def __rdiv__(self, other):
80          t=Complex(other,0)
81          return t/self

83      def __str__(self):
84          """Funci_n que genera una cadena de caracteres con la
85          representaci_n de un n_mero complejo
86          """
87          return "{:f}{:+f}i".format(self.re,self.im)
```

Una mejor sobrecarga de operadores

ComplexClassBetterOverloading.py parte 6

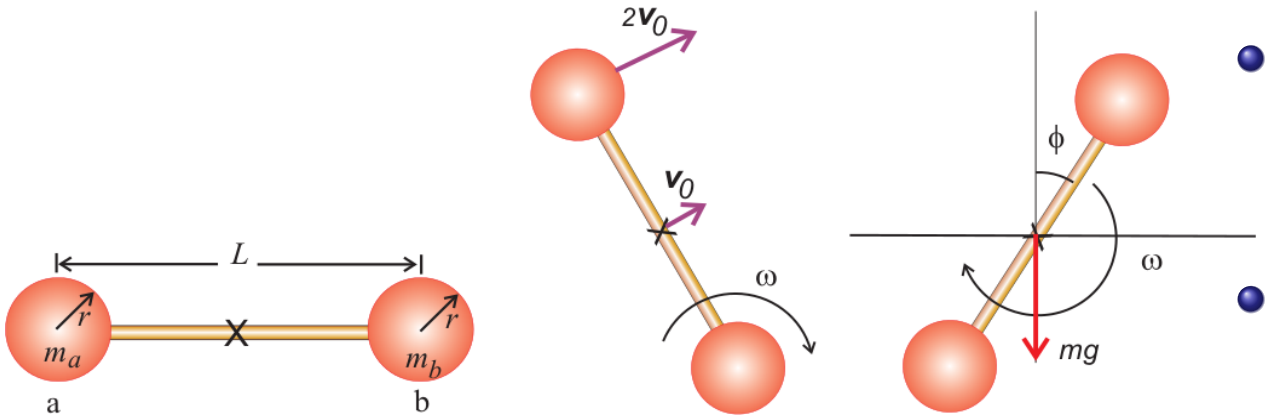
```
89 if __name__=="__main__":
90     q=Complex(1.,5.)
91     w=Complex(3.,4.)
92     h=Complex(4.)

94     a=q+w
95     a1=a*3
96     a3=-w-5*q
97     b=q*h
98     c=w/h

100    print a
101    print a1
102    print b
103    print c
```

Ejemplo de solución de un problema con POO

Trayectoria de un bastón de mando lanzado



Problemas a resolver:

- Movimiento parabólico del centro de masa
- Movimiento rotacional de las esferas al rededor del un punto en movimiento
- Ensamble de los 2 movimientos para resolver el problema completo

Movimiento parabólico del CM

Ver: path.py

Movimiento rotacional de 2 cuerpos alrededor de un punto en movimiento

Ver: rot.py

Solución del problema completo

Ver: baston.py