Creación de Complex class en Python

John Erick Cabrera Ramirez

Ejemplo inicial:

In [1]:
```
import numpy as np
# import inspect
# import threading
# from threading import Thread
```

In [2]:
```
class Persona:
    def __init__(self,nombre,apellido):#Definimos atributos de la clase
        self.nombre=nombre
        self.apellido=apellido
    def saludo(self): #una función para los objetos de la clase
        print(f"Bienvenido sr. {self.nombre} {self.apellido}")
```

In [3]:
```
p1=Persona("John","Cabrera")
print(p1.nombre)
print(p1.apellido)
p1.saludo()
```

```
John
Cabrera
Bienvenido sr. John Cabrera
```

In [4]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

Ejemplo 2: Clase que define los números complejos

Creación de la clase "Compleja":

In [5]:
```
class Compleja:
    #Atributos de un número complejo: Partes real e imaginaria
    def __init__(self, real, imaginaria):
        self.real = real
        self.imaginaria = imaginaria
    #definición de la suma, retorna un objeto complejo
    def suma(self,other,*args):
        a=self.real+other.real
        b=self.imaginaria+other.imaginaria
        return Compleja(a,b)
    #definición de la resta, retorna un objeto complejo
    def resta(self,other,*args):
        a=self.real-other.real
        b=self.imaginaria-other.imaginaria
        return Compleja(a,b)
    #definición del producto, retorna un objeto complejo
    #ver https://mathworld.wolfram.com/ComplexMultiplication.html
    def multiplicacion(self,other,*args):
        a=self.real*other.real-self.imaginaria*other.imaginaria
        b=self.real*other.imaginaria+self.imaginaria*other.real
        return Compleja(a,b)
    #módulo o norma de un número complejo
    def modulo(self,*args):
        return np.sqrt(self.real**2+self.imaginaria**2)
    #definición del cociente, retorna un objeto complejo
    #se calcula a partir de la norma al cuadrado del numero en el denominador
    #ver https://mathworld.wolfram.com/ComplexDivision.html
    def division(self,other,*args):
        a=(self.real*other.real+self.imaginaria*other.imaginaria)/(other.modulo()**2)
        b=(-self.real*other.imaginaria+self.imaginaria*other.real)/(other.modulo()**2)
        return Compleja(a,b)
    #Ingresa un objeto de la clase compleja
    #y retorna un string del número complejo escrito en la forma:
    #a+bi
    def imprimir(self,*args):
        if(self.imaginaria>=0):
            return f"{self.real}+{self.imaginaria}i"
        else:
            return f"{self.real}{self.imaginaria}i"
```

Llamado e impresión de todas las funciones/métodos. Se hace con este código "misterioso", para poder hacerlo sobre el programa de cualquiera de los estudiantes.

Tomado de https://stackoverflow.com/questions/37075680/run-all-functions-in-class

In [13]:
```
z1=Compleja(1,2)
z2=Compleja(3,4)
# z1=Compleja(5,4)
# z2=Compleja(3,2)
z3=Compleja(4,5)
print('z1=',z1.imprimir())
print('z2=',z2.imprimir())
print('z3=',z3.imprimir())
```

```
z1= 1+2i
z2= 3+4i
z3= 4+5i
```

In [14]:
```
#dir(objeto)  proporciona todos los métodos de la clase.
#getattr(objeto,metodo) evalúa el objeto en el método/function
method_names = [method for method in dir(z1)
                if callable(getattr(z1, method))
                if not method.startswith('_')]
#En algunos casos los métodos proporcionados por el docente o estudiantes tienen como sal
ida un string o float
#Se llama a la función según cada caso.
for method in method_names:
    if (type(getattr(z1, method)(z2))!=str)and(type(getattr(z1, method)(z2))!=np.float64
):
        print(f"z1 {method} z2=",(getattr(z1, method)(z2)).imprimir())# call
    else:
        print(f"{method} z1=",getattr(z1, method)(z2))
```

```
z1 division z2= 0.44+0.08i
imprimir z1= 1+2i
modulo z1= 2.23606797749979
z1 multiplicacion z2= -5+10i
z1 resta z2= -2-2i
z1 suma z2= 4+6i
```

In [8]:
```
method_names = [method for method in dir(z1)
                if callable(getattr(z1, method))
                if not method.startswith('_')]  # 'private' methods start from _
for method in method_names:
    if (type(getattr(z1, method)(z3))!=str)and(type(getattr(z1, method)(z3))!=np.float64
):
        print(f"z1 {method} z3=",(getattr(z1, method)(z3)).imprimir())# call
    else:
        print(f"{method} z1=",getattr(z1, method)(z3))
```

```
z1 division z3= 0.3414634146341463+0.07317073170731707i
imprimir z1= 1+2i
modulo z1= 2.23606797749979
z1 multiplicacion z3= -6+13i
z1 resta z3= -3-3i
z1 suma z3= 5+7i
```

Anexo

Algunas de las posibles notas para la sección 1

In [9]:
```
nota=0
cnt=0
while(nota<=50):
    print(cnt,nota)
    cnt+=1;
    if(nota<30):
        nota+=7.5
    else:
        nota+=5
```

```
0 0
1 7.5
2 15.0
3 22.5
4 30.0
5 35.0
6 40.0
7 45.0
8 50.0
```

Algunas de las posibles notas para la sección 2

In [10]:
```
nota=0
cnt=0
while(nota<=50):
    print(cnt,nota)
    cnt+=1;
    if(nota<30):
        nota+=6
    else:
        print("-")
        nota+=4
```

```
0 0
1 6
2 12
3 18
4 24
5 30
-
6 34
-
7 38
-
8 42
-
9 46
-
10 50
-
```

In [22]:
```
(40*2/5)/5
```

```
3.2
```

In [ ]:
```

```