

python



04



Ejemplo práctico

Aplicaciones de Escritorio con

Tkinter



Contenido

- Ejemplo práctico:
 - Operaciones con fracciones mixtas
- Funciones dentro de funciones
- Herencia: Reutilizando código de la clase padre
- Diseño de pantalla

The image shows a web application interface for performing operations on mixed fractions. It is titled 'Fraccion 1:' and 'Fraccion 2:'. Each fraction input area contains three fields: 'Ent' (integer part), 'Num' (numerator), and 'Den' (denominator). Below these, there is a dropdown menu for 'Operación:' with 'Suma' selected, and a 'Calcular' button. At the bottom, there is a 'Resultado:' label followed by a large text input field for the result.



Funciones dentro de funciones

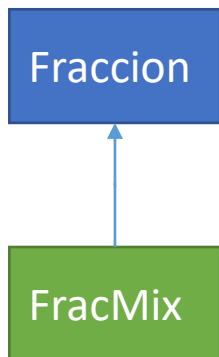
- Puedes declarar una función dentro de otra función
- La vida de la función interna es efímera al pertenecer al ámbito de la función donde es definida.
- La función interna va a ser creada de nuevo cada vez que llames a la función donde es creada y dejará de existir cuando esta termine. Esto implica que no podrá ser llamada desde fuera de la función donde es definida.

```
def simplifica(self):  
    d=self.mcd(self.num,self.den)  
    self.num=int(self.num/d)  
    self.den=int(self.den/d)  
  
def mcd(self,a,b):  
    if b==0:  
        return a  
    else:  
        return self.mcd(b,a%b)
```

```
def simplifica(self):  
  
    def mcd(a,b):  
        if b==0:  
            return a  
        else:  
            return mcd(b,a%b)
```

```
d=mcd(self.num,self.den)  
self.num=int(self.num/d)  
self.den=int(self.den/d)
```





Herencia

(Reutilizando código de la clase padre)

```

class Fraccion:
    def __init__(self, num=0, den=1): ...
    def __str__(self): ...
    def __mul__(self, obj): ...
    def __div__(self, obj): ...
    def __add__(self, obj): ...
    def __sub__(self, obj): ...
    def __eq__(self, b): ...
    def simplifica(self): ...
  
```

```

class FracMix(Fraccion):
    def __init__(self, ent, num=0, den=1):
        self.ent=ent
        super().__init__(num, den)
        self.simplifica()
        super().simplifica()

    def __str__(self):
        return str(self.ent) + super().__str__()

    def simplifica(self):
        if self.num > self.den:
            aux=self.num//self.den
            self.ent=self.ent+aux
            self.num-=(aux*self.den)
  
```

CONVERTIR FRACCIONES MIXTAS A IMPROPIAS

$$3 \frac{1}{5} \xrightarrow{x} \frac{16}{5}$$

CONVERTIR FRACCIONES IMPROPIAS A MIXTAS

$\frac{7}{3} = 7 > 3$ fracción impropia

$$\frac{7}{3} = 2 \frac{1}{3}$$

fracción mixta

$$2 \frac{1}{3}$$

parte entera

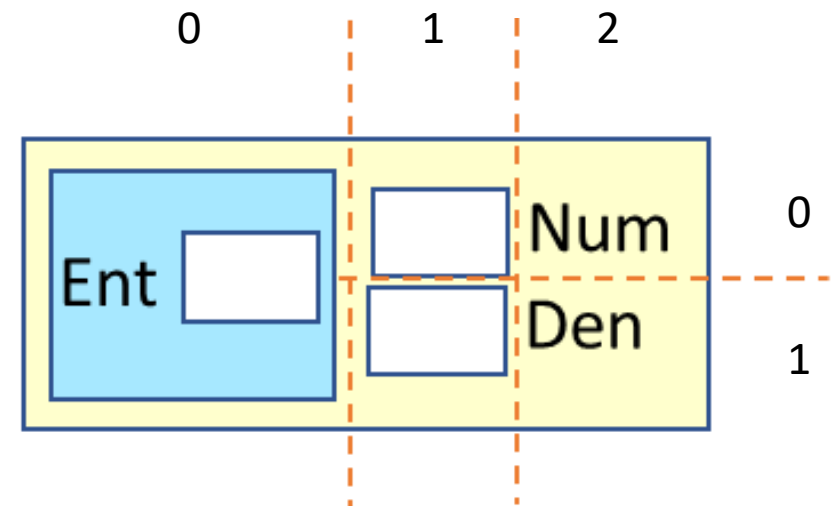
parte fraccionaria

Operator	Method	Expression
+ Addition	<code>__add__(self, other)</code>	<code>a1 + a2</code>
- Subtraction	<code>__sub__(self, other)</code>	<code>a1 - a2</code>
* Multiplication	<code>__mul__(self, other)</code>	<code>a1 * a2</code>
@ Matrix Multiplication	<code>__matmul__(self, other)</code>	<code>a1 @ a2 (Python 3.5)</code>
/ Division	<code>__div__(self, other)</code>	<code>a1 / a2 (Python 2 only)</code>
/ Division	<code>__truediv__(self, other)</code>	<code>a1 / a2 (Python 3)</code>
// Floor Division	<code>__floordiv__(self, other)</code>	<code>a1 // a2</code>
% Modulo/Remainder	<code>__mod__(self, other)</code>	<code>a1 % a2</code>
** Power	<code>__pow__(self, other[, modulo])</code>	<code>a1 ** a2</code>
<< Bitwise Left Shift	<code>__lshift__(self, other)</code>	<code>a1 << a2</code>
>> Bitwise Right Shift	<code>__rshift__(self, other)</code>	<code>a1 >> a2</code>
& Bitwise AND	<code>__and__(self, other)</code>	<code>a1 & a2</code>
^ Bitwise XOR	<code>__xor__(self, other)</code>	<code>a1 ^ a2</code>
(Bitwise OR)	<code>__or__(self, other)</code>	<code>a1 a2</code>
- Negation (Arithmetic)	<code>__neg__(self)</code>	<code>-a1</code>
+ Positive	<code>__pos__(self)</code>	<code>+a1</code>
~ Bitwise NOT	<code>__invert__(self)</code>	<code>~a1</code>
< Less than	<code>__lt__(self, other)</code>	<code>a1 < a2</code>
<= Less than or Equal to	<code>__le__(self, other)</code>	<code>a1 <= a2</code>
== Equal to	<code>__eq__(self, other)</code>	<code>a1 == a2</code>
!= Not Equal to	<code>__ne__(self, other)</code>	<code>a1 != a2</code>
> Greater than	<code>__gt__(self, other)</code>	<code>a1 > a2</code>
>= Greater than or Equal to	<code>__ge__(self, other)</code>	<code>a1 >= a2</code>
[index] Index operator	<code>__getitem__(self, index)</code>	<code>a1[index]</code>
in In operator	<code>__contains__(self, other)</code>	<code>a2 in a1</code>
(*args, ...) Calling	<code>__call__(self, *args, **kwargs)</code>	<code>a1(*args, **kwargs)</code>



Diseño de pantalla

Fraccion 1:		Fraccion 2:	
Ent	<input type="text"/>	Ent	<input type="text"/>
	<input type="text"/>		<input type="text"/>
	<input type="text"/>		<input type="text"/>
	Num		Num
	Den		Den
Operación:	Suma		
	<input type="button" value="▼"/>		
			<input type="button" value="Calcular"/>
Resultado:	<input type="text"/>		



python



04



Ejemplo práctico

Aplicaciones de Escritorio con

Tkinter

