Programación II

Unidad 0 - repaso: Programación Orientada a Objetos

Universidad Nacional de Rosario.
Facultad de Ciencias Exactas, Ingeniería y Agrimensura.

Repaso

Repasaremos los conceptos fundamentales de la POO desarrollando un conjunto de objetos que nos permitan representar elementos de geometría analítica en dos dimensiones.
El concepto fundamental será el de Punto, el cual describe una posición en el plano determinada por sus coordenadas cartesianas.

Definición de clases

Comenzamos definiendo la clase y su método constructor

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Instanciando objetos

Podemos crear objetos pertenecientes a la clase del siguiente modo

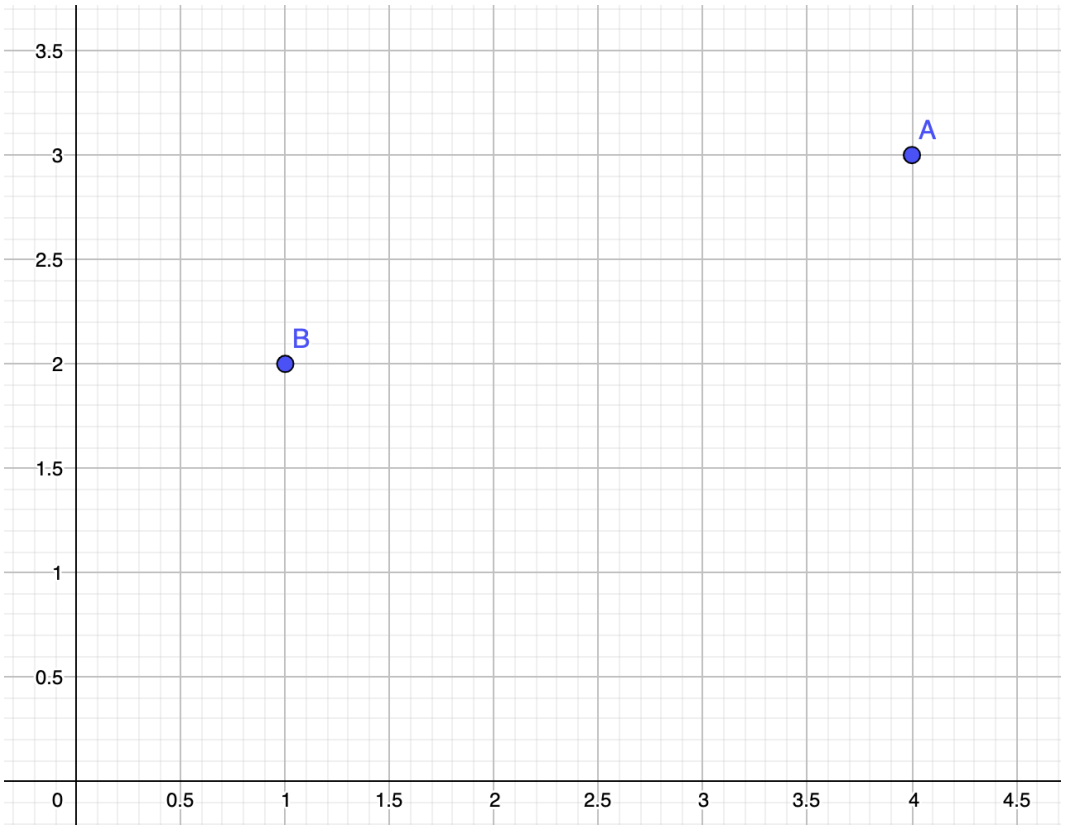
```
class Point:
    ...

A = Point(4, 3)
B = Point(1, 2)
```

Cuando llamamos a `Point`, Python se encarga de llamar al método constructor por nosotros, pasándole los argumentos que corresponden y nos devuelve un nuevo objeto.

Decimos que `__init__` es un *método mágico* porque no lo invocamos explícitamente, si no que Python lo hace por nosotros cuando creamos un nuevo objeto.

Programación orientada a objetos



Atributos

Las coordenadas `x` e `y` son *atributos* de los objetos de tipo Point.
Podemos obtener el valor del atributo en un objeto utilizando la notación punto.

```
class Point:
    ...

A = (4, 3)
print(A.x) # Imprime 4
print(A.y) # Imprime 3
```

De la misma forma podemos modificar los atributos de un objeto.

Imprimiendo Objetos

Para poder imprimir de forma amigable objetos, debemos definir el método mágico `__str__`.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f"({self.x}, {self.y})"

A = Point(4, 3)
print(A) # Imprime (4, 3)
```

De nuevo `__str__` es un método mágico porque no lo llamamos explícitamente.
Python lo hace por nosotros cuando le pedimos mostrar el objeto.

Igualdad de Objetos

Dos objetos distintos pueden representar el mismo valor, (así como las expresiones 5, 25/5 y 5.0 son todas representaciones del "valor"5).
Para que Python pueda decidir si dos objetos representan el mismo valor, implementamos el método mágico __eq__.

```
class Point:
    ...

    def __eq__(self, other):
        return self.x == other.x and \
           self.y == other.y

A = Point(4, 3)
B = Point(1, 2)
C = Point(1, 2)
print(A == B) # Falso
print(B == C) # Verdadero
```

Métodos mágicos

Los métodos mágicos son aquellos métodos de un objeto que son llamados por Python sin que nosotros lo hagamos explícitamente.
Los métodos `__init__`, `__str__` y `__eq__` son sólo algunos ejemplos, en la documentación de Python tenemos una referencia completa de los métodos mágicos, como se definen y la manera apropiada de hacerlo. [1]
Otro ejemplo de método mágico es `__sub__` el cual nos permitirá implementar la *diferencia* entre dos puntos, entendida como la distancia euclideana entre ellos.

[1] https://docs.python.org/3/reference/datamodel.html#special-method-names

Métodos mágicos

```
from math import dist

class Point:
    ...
    def __sub__(self, other):
        return dist(
            (self.x, self.y),
            (other.x, other.y),
        )

A = Point(1, 2)
B = Point(3, 2)
print(A-B) # Imprime 2
```

Composición de clases

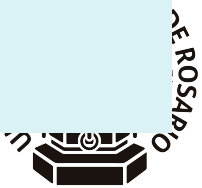
Ya estamos en condiciones de empezar a representar objetos más complejos, por ejemplo, polígonos. Un polígono queda definido a partir de sus vértices. Es decir, la características o atributos que describen un rectángulo son objetos de tipo Point. A esta práctica, de emplear clases dentro de la definición de otra, la llamamos **composición de clases**.

Composición de clases

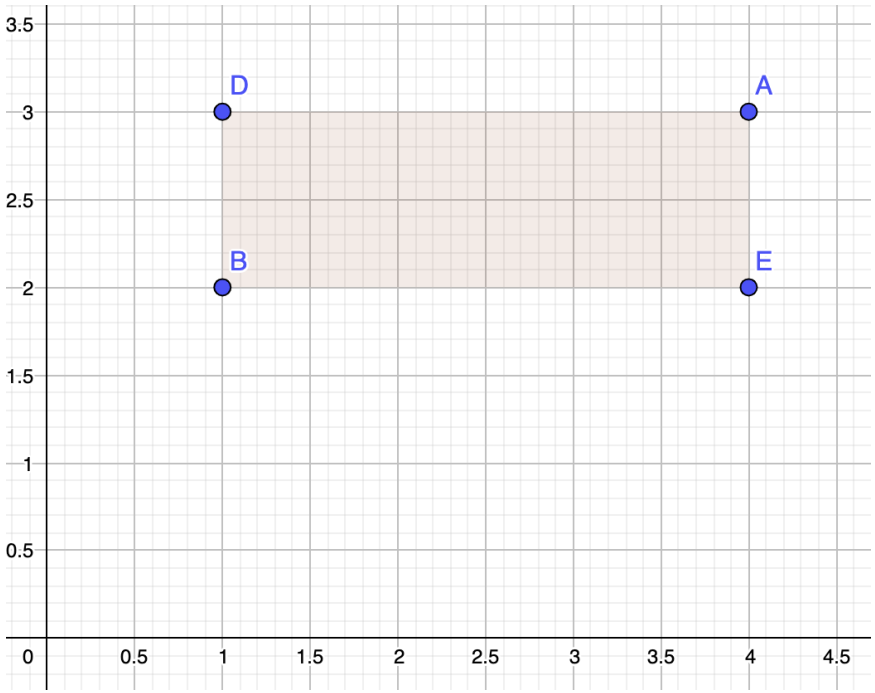
```
class Point:
    ...

class Polygon:
    def __init__(self, vertices):
        self.vertices = vertices

# Primero instanciamos los puntos
A = Point(4, 3)
B = Point(1, 2)
D = Point(1, 3)
E = Point(4, 2)
#Construimos el rectángulo
rectangulo = Polygon([A, E, B, D])
```



Composición de clases



Herencia

Supongamos que nuestra aplicación de geometría analítica ahora necesita permitir el cálculo del área de un rectángulo.
Debemos definir una clase Rectángulo que represente a estos objetos, pero como los rectángulos son casos particulares de los polígonos, podemos aprovechar el código que ya tenemos escrito y solo agregar o cambiar la funcionalidad que sea necesaria.
A esto lo llamamos **herencia de clases**.

Herencia

```
class Rectangle(Polygon):
    def area(self):
        A = self.vertices[0]
        E = self.vertices[1]
        B = self.vertices[2]
        D = self.vertices[4]
        base = B - E
        altura = B - D
        return base * altura

A = Point(4, 3)
B = Point(1, 2)
D = Point(1, 3)
E = Point(4, 2)
rectangulo = Polygon([A, E, B, D])
print(rectangulo.area()) # Imprime 3
```

¿PREGUNTAS?

Referencias

Apunte de Cátedra

Elaborados por el staff docente.

Será subido al campus virtual de la materia.

A. Downey et al, 2002.

How to Think Like a Computer Scientist. Learning with Python.

Capítulos 12 a 16