Introducción a la computación

1er cuatrimestre de 2016

Implementación de TADs

TAD Punto

CrearPunto(x,y) → Punto: Crea un nuevo punto

P.CoordenadaX() → $\mathbb{R}$: Devuelve la coordenada x de P

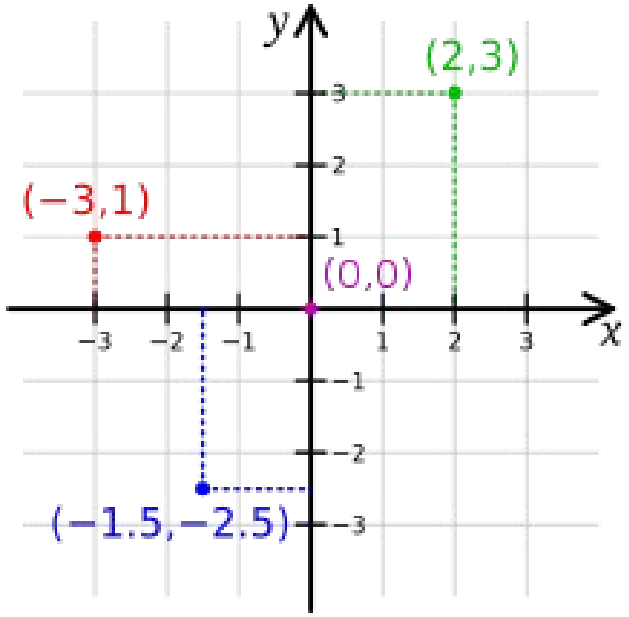
P.CoordenadaY() → $\mathbb{R}$: Devuelve la coordenada y de P

P.DistanciaAlOrigen() → $\mathbb{R}$:Devuelve la distancia al origen

Dónde P:Punto y x,y: $\mathbb{R}$

Implementación de TADs

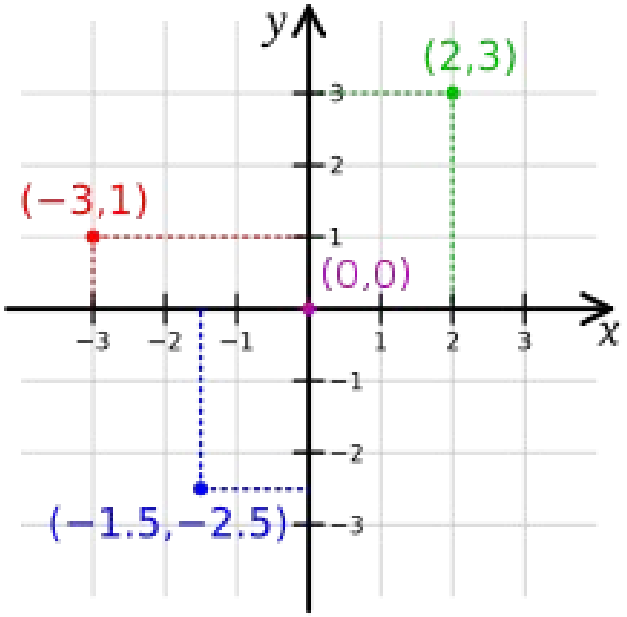
TAD Punto




```
>>> x1 = -3
>>> y1 = -1
>>> x2 = 0
>>> y2 = 0
>>> x3 = 2
>>> y3 = 3
>>> x4 = -1.5
>>> y4 = -2.5
```

Implementación de TADs

TAD Punto

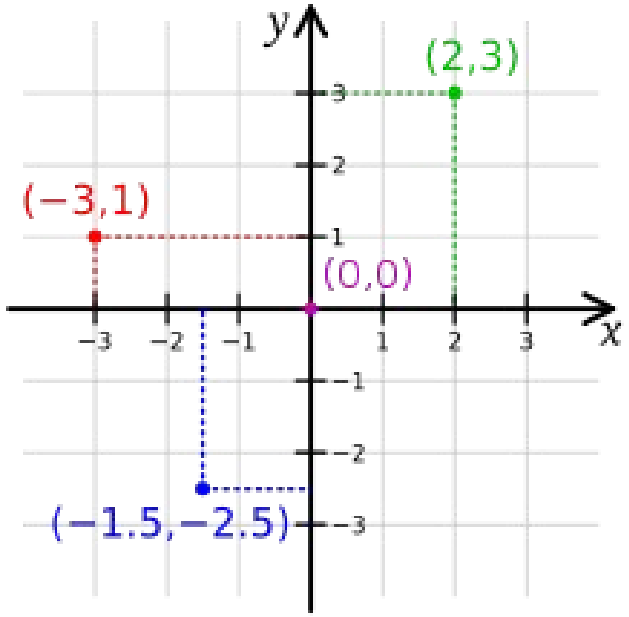


Y si estuvieramos en $\mathbb{R}^7$?

```
>>> x1 = -3
>>> y1 = -1
>>> x2 = 0
>>> y2 = 0
>>> x3 = 2
>>> y3 = 3
>>> x4 = -1.5
>>> y4 = -2.5
```

Implementación de TADs

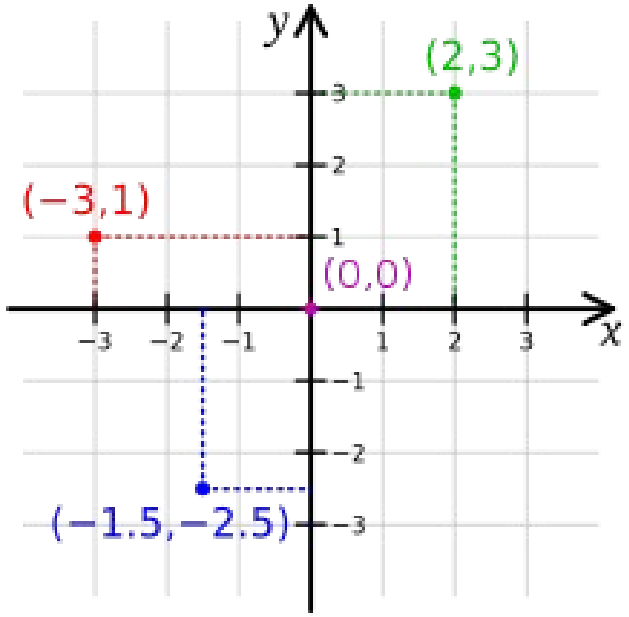
TAD Punto




```
>>> p1 = [-3,1]
>>> p2 = [0,0]
>>> p3 = [2,3]
>>> p4 = [-1.5, -2.5]
```

Implementación de TADs

TAD Punto



Las listas no son puntos!

```
>>> p1 = [-3,1]
>>> p2 = [0,0]
>>> p3 = [2,3]
>>> p4 = [-1.5, -2.5]
>>> p5 = [8, 3]
>>> p5.sort()
>>> p5
[3,8]
```

Implementación de TADs

Programación estructurada:

- funciones y procedimientos;
- tipos de datos: enteros, reales, strings, arreglos.

Programación orientada a objetos:

- modela entes del mundo, sus propiedades y su comportamiento;
- Python es un lenguaje orientado a objetos.

Implementación de TADs

TAD Punto

CrearPunto(x,y) → Punto: Crea un nuevo punto

P.CoordenadaX() → $\mathbb{R}$: Devuelve la coordenada x de P

P.CoordenadaY() → $\mathbb{R}$: Devuelve la coordenada y de P

P.DistanciaAlOrigen() → $\mathbb{R}$:Devuelve la distancia al origen

Estructura de representación:

`Tipo Punto == <x:`$\mathbb{R}$`, y:`$\mathbb{R}$`>`

Implementación de TADs

Clases

```
class Punto:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def distanciaAlOrigen(self):
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5

p = Punto(2,3)
print p.distanciaAlOrigen()
```

Implementación de TADs

Instancias

```
class Punto:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def distanciaAlOrigen(self):
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5

p = Punto(2,3)
print p.distanciaAlOrigen()
```

Implementación de TADs

Constructor

```
class Punto:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def distanciaAlOrigen(self):
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5

p = Punto(2,3)
print p.distanciaAlOrigen()
```

Implementación de TADs

Self

```
class Punto:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def distanciaAlOrigen(self):
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5

p = Punto(2,3)
print p.distanciaAlOrigen()
```

Implementación de TADs

Propiedades

```
class Punto:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def distanciaAlOrigen(self):
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5

p = Punto(2,3)
print p.distanciaAlOrigen()
```

Implementación de TADs

Valores default

```
class Punto:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def distanciaAlOrigen(self):
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5

p = Punto(2,3)
print p.distanciaAlOrigen()
q = Punto() # es el punto (0,0)
```

Implementación de TADs

Métodos

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def distanciaAlOrigen(self):
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5

p = Point(2,3)
print p.distanciaAlOrigen()
```

Implementación de TADs

Funciones que usan la clase

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
            ...

def puntoMedio(p1, p2):
    x = (p1.coordenadaX() + p2.coordenadaX())/2.0
    y = (p1.coordenadaY() + p2.coordenadaY())/2.0
    return Punto(x,y)

p = Punto(0,0)
q = Punto(2,1)
r = puntoMedio(p,q) # r es el punto (1,0.5)
```

Implementación de TADs

Funciones que usan la clase

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
            ...

def puntoMedio(p1, p2):
    x = (p1.coordenadaX() + p2.coordenadaX())/2.0
    y = (p1.coordenadaY() + p2.coordenadaY())/2.0
    return Punto(x,y)

p = Punto(0,0)
q = Punto(2,1)
r = puntoMedio(p,q) # r es el punto (1,0.5)
```

Implementación de TADs

Igualdad observacional

```
class Punto:

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

    def __str__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

p = Punto(2,3)
q = Punto()
r = Punto(0,0)
print p==q, q==r
```

Implementación de TADs

Igualdad observacional

```
class Punto:

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

    def __str__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

p = Punto(2,3)
q = Punto()
r = Punto(0,0)
print p==q, q==r # imprime False, True
```

Implementación de TADs

STR

```
class Punto:

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

    def __str__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

p = Point(2,3)
print p # imprime (2,3)
```

Implementación de TADs

Otros operadores

```
def __lt__(self, other): # p1 < p2
...
def __le__(self, other): # p1 <= p2
...
def __eq__(self, other): # p1 == p2
...
def __ne__(self, other): # p1 != p2
...
def __gt__(self, other): # p1 > p2
...
def __ge__(self, other): # p1 >= p2
...
```

Implementación de TADs

Ejercicio. Implementar en Python el TAD Fecha

CrearFecha(día,mes,año) → Fecha: crea una nueva Fecha

F.ObtenerDia() → $\mathbb{Z}$: Devuelve el día de F

F.ObtenerMes() → $\mathbb{Z}$: Devuelve el mes de F

F.ObtenerAño() → $\mathbb{Z}$: Devuelve el año de F

F.EsAnterior(F_2) → Bool: Devuelve True si F es anterior a F_2 y False en caso contrario

F.Siguiente() → Fecha : Dev. la fecha inmediata posterior