PYTHON

UNI – FIIS
Glen Rodríguez R.

Trabajando con archivos en Python

- Hay que asociar una variable con un archivo en disco (“manejador del archivo”)
- Si queremos leer el archivo, hay que posicionar el “puntero del file” a algún punto (por ejemplo, al inicio del archivo)

Formato:

```
<variable manejadora> = open(<nombre de archivo>, "r")
```

Ejemplo:

(si el archivo se conoce de antemano)

```
inputFile = open("data.txt", "r")
```

O

(si se debe preguntar al usuario el nombre del archivo)

```
filename = input("Entre nombre del archivo a leer: ")
inputFile = open(filename, "r")
```

Posicionando el puntero

- Por default: en el inicio del archivo
- Para moverlo: <var.manejadora>.seek(offset, base)
- Offset puede ser positivo, negativo o cero
- Base por default=0 (inicio de archivo), 1 (posición actual). 2 (fin del archivo)
- Posición nueva: “base”+offset

Leyendo texto del archivo

- Generalmente se lee todo un archivo, línea por línea, con un bucle

Formato:

```
for <variable string> in <manejador del archivo>:
    <Leer texto>
```

Ejemplo:

```
for line in inputFile:
    print(line)  # imprime el contenido
```

Cerrando el archivo

- Cerrar= dejar de tenerlo “separado” y liberarlo para que otro programa lo use
- No es obligatorio (el archivo se cierra cuando acaba el programa) pero es buena práctica.

Formato:

```
<var. manejador>.close()
```

Ejemplo:

```
inputFile.close()
```

Ejemplo

```
inputFileName = input("Entre nombre del archivo: ")
inputFile = open(inputFileName, "r")
print("Abriendo archivo ", inputFileName, " para lectura.")

for line in inputFile:
    sys.stdout.write(line)

inputFile.close()
print("Se terminó de leer el archivo ", inputFileName)
```

Escribiendo datos a un archivo

- Hay que abrirlo pero “para escritura” (se reserva su uso)

Formato:

```
<var. manejador> = open(<nombre del archivo>, "w")
```

Ejemplo:

(si el archivo se conoce de antemano)

```
outputFile = open("notas.txt", "w")
```

O

(si se debe preguntar al usuario el nombre del archivo)

```
filename2 = input("Entre nombre del archivo de notas: ")
outputFile = open(filename2, "r")
```

Escribiendo

- Use la función write( )
- Solo se escriben strings. Si desea escribir números, conviertalos a string

Formato:

```
outputFile.write(temp)
```

Ejemplo:

```
# Asumir que temp contiene un string.
outputFile.write (temp)
```

Ejemplo

```
inputFileName = input("Nombre de archivos con las notas? ")
outputFileName = input("Nombre del archivo para grabar promedios: ")

inputFile = open(inputFileName, "r")
outputFile = open(outputFileName, "w")

print("Abriendo archivo ", inputFileName, " para leer.")
print("Abriendo archivo ", outputFileName, " para escribir.")
prom = 0
```

Ejemplo

```
for line in inputFile:
    alumno, n1,n2,n3 = line.split(“,”)
    temp = alumno + “ ”
    temp = temp + str ((n1+n2+n3))
    temp = temp + '\n'
    print (alumno)
    outputFile.write (temp)
inputFile.close ()
outputFile.close ()
print ("Termine de leer ", inputFileName)
print ("Termine de escribir ", outputFileName)
```

En vez del for se puede usar un while

```
inputFile = open(inputFileName, "r")

……

……


line = inputFile.readline()


while (line != ""):
    # procesar la linea leida antes
    line = inputFile.readline()   #leer sgte linea
```

Excepciones

- Errores detectados en tiempo de ejecución (tratar de abrir un archivo que no existe, dividir entre cero, etc.)
- Si no se manejan, harían colgar el programa
- Se pueden “manejar” o administrar para evitar la caída del programa

Estructura básica

```
try:
    instrucción que podría causar una excepción
except <tipo de excepicón>:
    Reaccionar ante el error
else:  # No siempre se necesita el else
    proceso normal (si no hubo error)
finally:  # No siempre se necesita
    Acciones que siempre deben ejecutarse
```

Ejemplo: con archivos

```
inputFileOK = False
while (inputFileOK == False):
   try:
      inputFileName = input("Entre nombre de archivo: ")
      inputFile = open(inputFileName, "r")
   except IOError:
      print("Archivo ", inputFileName, " no se pudo abrir")
   else:
      print("Abriendo archivo ", inputFileName, " para leer.")
      inputFileOK = True

      for line in inputFile:
         sys.stdout.write(line)
      print ("Terminamos de leer ", inputFileName)
      inputFile.close()
      print ("Cerrando archivo ", inputFileName)
```

Continuación

```
    # aún dentro del while
finally:
    if (inputFileOK == True):
        print ("Se pudo leer el archivo ", inputFileName)
    else:
        print ("No se pudo leer ", inputFileName)
```

Excepción de ingreso de datos

```
inputOK = False
while (inputOK == False):
    try:
        num = input("Entre un número: ")
        num = float(num)
    except ValueError:    # No se pudo convertir a número
        print("Data no numérica recibida: '%s'" %num)
    else:   # si se pudo
        inputOK = True
num = num * 2
print(num)
```

Excepciones populares

- EOFError: se intentó leera archivo sin datos
- ImportError: problemas al tratar de importar un modulo
- ModuleNotFoundError: modulo no existe
- IndexError: indice excede tamaño de la secuencia
- KeyError: no hay una clave en el diccionario
- KeyboardInterrupt: usuario aprieta Control-C

Excepciones populares

- MemoryError: sistema sin memoria libre
- OSError: error del sistema operativo
- TypeError: se uso un tipo de datos que no se esperaba
- ZeroDivisionError
- ValueError: logaritmo de un negativo, sqrt de un negativo, int("hola")
- RuntimeError: otros errores

Clases

- Una clase es la especificación de una plantilla genérica para una variable compuesta heterogénea
- Permite crear variables compuestas más complicadas que listas y diccionarios
- Solo define como un caso concreto (instancia) sería, pero no lo crea.
- Lo más importan es definir que datos (atributos) tienen los objetos

Nombre:
Teléfono:
Email:
Compras:

Nombre:
Teléfono:
Email:
Compras:

Nombre:
Teléfono:
Email:
Compras:

Definiendo una clase

Formato:

```
class <Nombre de la clase>:  #costumbre: empieza en mayúscula
     nombre del 1er campo = <valor por default>
     nombre del 2do campo = <valor por default>
```

Ejemplo:

```
class Cliente:
    nombre = "default"
    telefono = "(123)456-7890"
    email = "foo@bar.com"
    compras = 0
```

Instanciando un objeto

- Crear una instancia (objeto) de una clase se llama instanciación

Format:

```
<nombre de “variable”> = <nombre de la clase>()
```

Example:

```
primerCliente = Cliente()
```

Definir la case vs. Crear una instancia de esa clase

- Definiendo clase
  - Una plantilla que describe a una clase: cuantos campos tiene, de que tipos,que valo por default tienen.
- Creando objeto
  - Instancias de la clase pueden tomar diferentes formas.

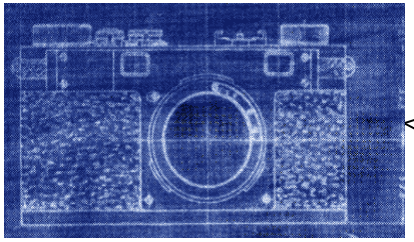


Image copyright unknown

Accesando atributos (campos)

Format:

```
<objeto>.<nombre del campo>                # Leyendo
<objeto>.<nombre del campo> = <valor>      # Modificando
```

Example:

```
primerCliente.telefono = "1239900"
```

Ejemplo de usar objeto en vez de lista

```
Lista:

miCliente=[“James Tam”,”(123)456-7890”,”tam@uni.edu.pe”,0]


class Cliente:

    nombre = "default"

    telefono = "(123)456-7890"

    email = "foo@bar.com"

    compras = 0
```

Continuación

```
def main():
    primerCliente = Cliente()
    primerCliente.name = "James Tam"
    primerCliente.email = "tam@uni.edu.pe"
    print(primerCliente.nombre)
    print(primerCliente.telefono)
    print(primerCliente.email)
    print(primerCliente.compras)


main()
```

Diferencias?

- Necesito crear la lista en el orden exacto. No debo preocuparme del orden en el objeto
- Valores por default (campo: teléfono) en el objeto no necesitan ser inicializados a mano
- Qué pasa se quiero incluir fax después de teléfono? Debo alterar las listas

```
[“James Tam”,”(123)456-7890”,”NUEVO DATO DE FAX”,”tam@uni.edu.pe”,0]
```

Beneficios

- Permite crear nuevos tipos de variable
- Ese tipo puede modelar cualquier entidad
- Entidad? Qué es eso?
- Es algo con existencia individual, identificable, independiente / separable de otros “algos”.

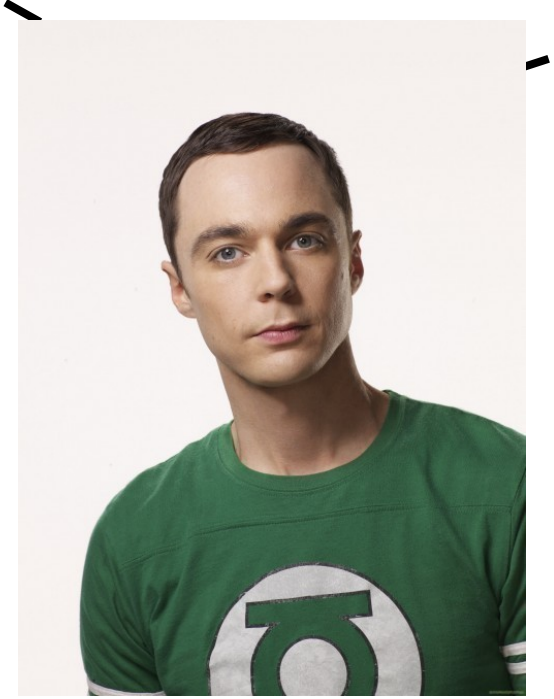
Beneficios

- Nombre para los campos, no orden
- Solo los campos predefinidos son aceptados
  - PrimerCliente.apellido=”Perez” → error
- Deben ir “si o si”
- Pueden usarse como “ladrillos” para construir objetos más complejos

Las clases también tienen comportamientos

ATRIBUTOS
Nombre:
Teléfono:
Email:
Compras:

COMPORTAM.
Abrir cuenta
Reservar prod.
Hacer pedido
Cancelar pedido



Comportamientos (métodos)

- Son funciones que están amarradas a una clase, no existen independientemente, si no que existen a través de la instancia de una clase (un objeto). Ej.:
  - nombre, extension = filename.split(“.”)
- Tú las defines
- Más conocidas como METODOS

Definiendo métodos

Formato:

```
class <nombre de clase>:
    def <nombre de método> (self, <otros parámetros>):
        <cuerpo>
```

Ejemplo:

```
class Persona:
   nombre = "Juanito Alimaña"
   def presentate (self):
      print ("Me llamo ", self.nombre)
```

Cuando un atributo debe ser accesado en un método, debe usar el sufijo “self”

Ejemplo

```
class Persona:
   nombre = "Juanito alimaña"
   def presentate(self):
      print("Me llamo ", self.nombre)


def main():
   aPersona = Persona()
   aPersona.presentate()
   aPersona.nombre = "David Banner"
   aPersona.presentate()


main()
```

“Self”

- Loes métodos deben tener por lo menos 1 parámetro, el “self”
- Se refiere al mismo objeto
- Sirve para distingir atributos de diferente objetos

```
bart = Persona()

lisa = Persona()

lisa.presentate()
```

```
def presentate():
    print "Me llamo", nombre
```

Nombre de quien?

Ejemplo

```
   class Persona:
      nombre = "Juanito alimaña"
      def presentate(self):
         print("Me llamo ", self.nombre)

def main():
   lisa = Persona()
   lisa.nombre = "Lisa Simpson, la lista"
   bart = Persona()
   bart.name = "Bart Simpson, el Barto"

   lisa.sayName()
   bart.sayName()

main()
```

Inicializando atributos

- Método constructor. Se llama automáticamente cuando se crea un objeto

Formato:

```
class <nombre de la clase>:
def __init__(self, <otros parámetros>):
     <cuerpo>
```

Ejemplo:

```
class Person:
   nombre = ""
   def __init__(self):
       self.nombre = "Desconocido"
```

Inicialización no estándar

```
class Persona
    nombre = "Default"
    def __init___(self, aName):
        self.nombre = aName
```

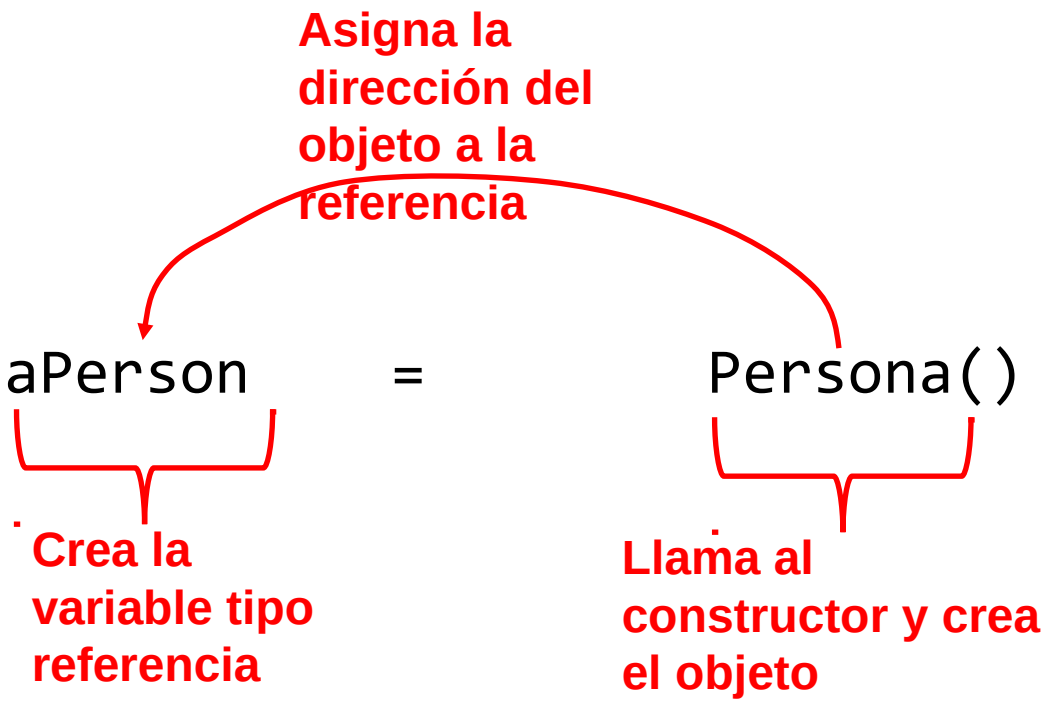
O

```
class Persona
    def __init___(self, aName):
        self.nombre = aName
```

LLAMADA

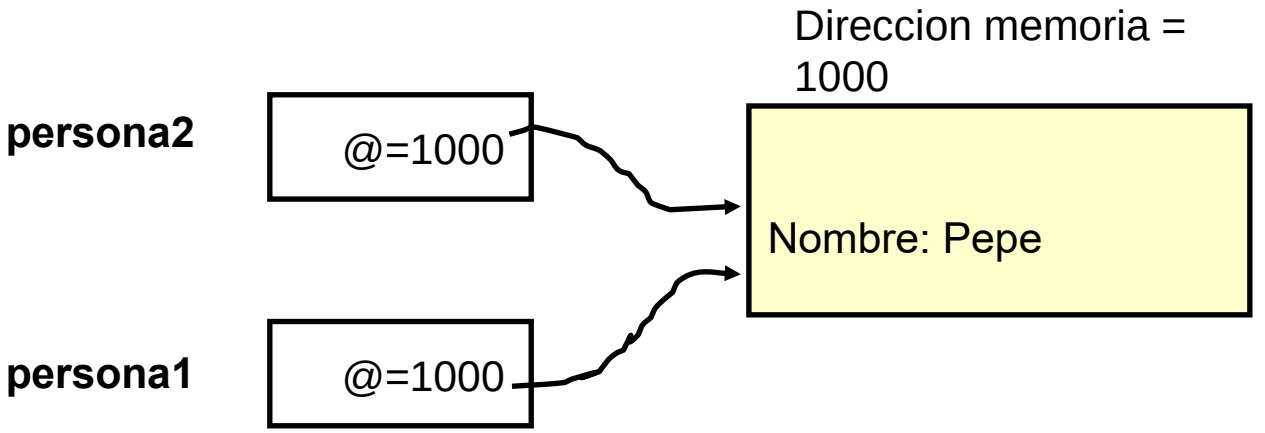
```
aPerson = Persona("Vladimir Putin")
```

Los objetos son direcciones, como las listas



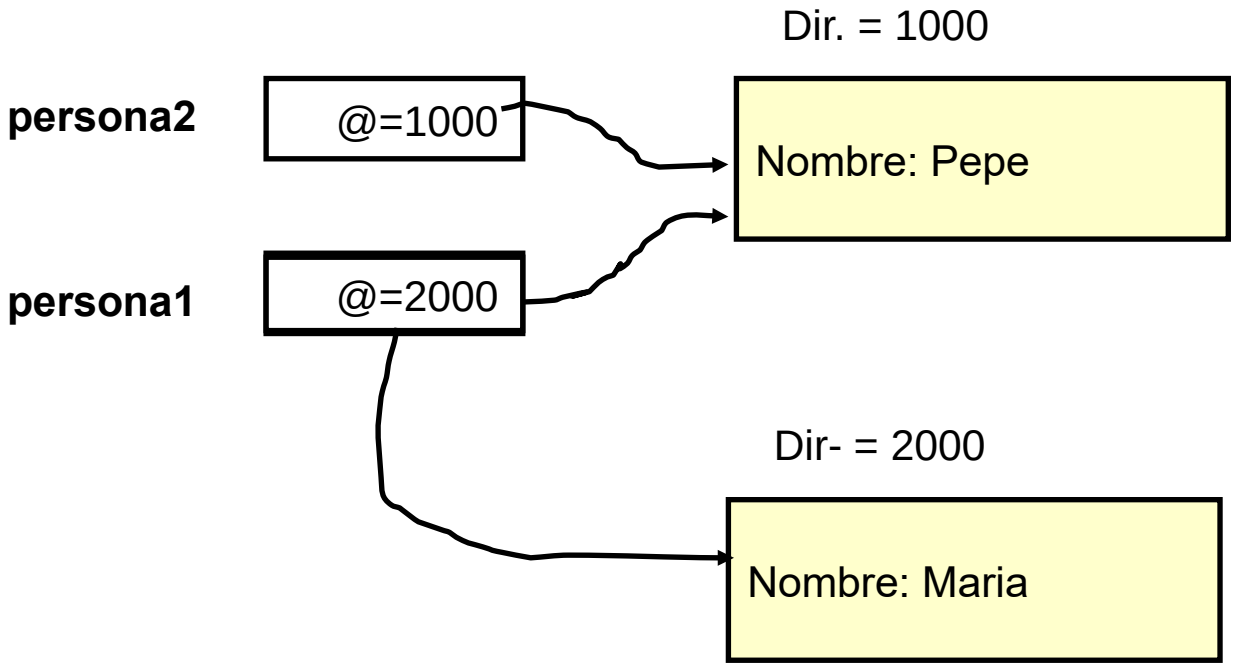
Analizar !

```
persona1 = Persona("Pepe")

persona2 = persona1

muestraNombre(persona1)

muestraNombre(persona2)

print()
```



Analizar !

```
persona1 = Persona("Pepe")

persona2 = persona1

muestraNombre(persona1)

muestraNombre(persona2)

print()


persona1=Persona(“Maria”)

muestraNombre(persona1)

muestraNombre(persona2)
```



Valores por defecto

- En el constructor:

```
def __init__ (self, name = "Anonimo"):
    self.nombre = name


smiley = Persona()
print(smiley.nombre)
jt = Person("Jaime")
print(jt.nombre)
```

Clases en módulos

Generalmente las clases van en su propio módulo.

Costumbre: Que el nombre del modulo (archivo) sea igual al de la clase.

Archivo: Persona.py

```
class Persona:
     def fun1(self):
        print(“fun1”)

     def fun2 (self):
        print(“fun2”)
```

Paa usar la clase Persona en otro programa, hay que usar un import:

```
from <filename> import <class name>

from Persona import Persona
```

“self” y métodos

- Un método que usa otro método de la misma clase debe ser precedido por “self”

Ejemplo:

```
class Bar:
     x = 1
     def fun1(self):
         print(self.x)  # accesando atributo ‘x’


    def fun2(self):
        self.fun1()     # llamando al otro método ‘fun1’
```

Nombre del modulo (clase) principal

- Así cpomo se acostumbraba llamar “start” o “main” al “casi prograam principal”, hay que llamar a la clase principal (la que tiene el main o start) con un nombre reconocible
- Ejemplo: Driver, Game, Principal

Herencia

- La herencia es un mecanismo de la programación orientada a objetos que sirve para crear clases nuevas a partir de clases preexistentes. Se toman (heredan) atributos y comportamientos de las clases viejas y se los modifica para modelar una nueva situación.
- La clase vieja se llama clase base y la que se construye a partir de ella es una clase derivada.

Ejemplo: heredar de Persona

```
class AlumnoFIIS(Persona):
    def __init__(self, codigo):
        # Constructor de AlumnoFIIS
        # llamamos al constructor de Persona
        Persona.__init__(self, nombre)
        # agregamos el nuevo atributo
        self.codigo= codigo
    def __repr__(self):
        z="<"+self.nombre+">"
        return z
```

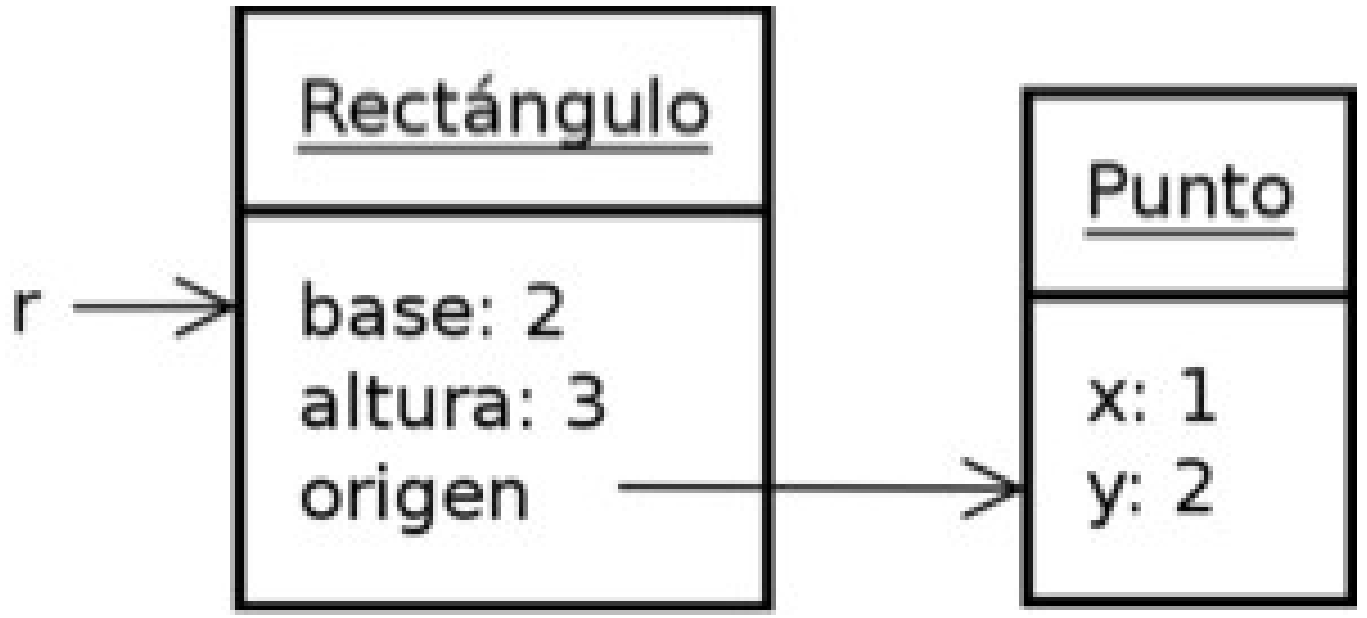
Probando

```
a = AlumnoFIUBA("DNI 35123456", "Damien", "Thorn", "98765")

print(a)
```

Delegación

- Cuando un atributo de un objeto es otro objeto
- Suponiendo que ya hay una clase Punto

```
class Circulo():


    def __init__(self, centro, radio):
        self.radio = radio
        self.centro = centro
CREACION:
c = Circulo(Punto(1, 2) , 8)
```


Rectángulo
base: 2
altura: 3
origen
r
Punto
x: 1
y: 2