

Parte II: Computación científica

Clase 13: Creando tipos de datos

Diego Caro
dcaro@udd.cl



Basada en presentaciones oficiales de libro Introduction to Programming in Python (Sedgewick, Wayne, Dondero).

Disponible en <https://introcs.cs.princeton.edu/python>

Creando nuevos tipos de datos

- Tipos de datos: conjunto de valores y operaciones sobre esos valores.
- Hoy: crearemos nuestros propios tipos de datos y operaciones sobre ellos.



Tipo de dato	Conjunto de valores	Operaciones
bool	true, false	! (not), (or), && (and)
int	0, 1, -1, 2, -2, 3, -3,....	+, -, /, *, %
double	0.001, 2.33, 3.1416	+, -, /, *
str	secuencia de caracteres	concatenar, comparar, etc...

Definiendo tipos de datos en Python

- Una **clase** (class) en Python permite crear tipos de datos, especificando:
 - Conjunto de valores (**atributos** o variables)
 - **Métodos** (funciones sobre los valores)
 - **Constructores** (función para crear e inicializar una clase)

The diagram illustrates the components of a Python class definition. It features a code block with the following content:

```
class Rectangle:
    def __init__(self, width, height):
        self.w = width
        self.h = height

    def area(self):
        return self.w * self.h

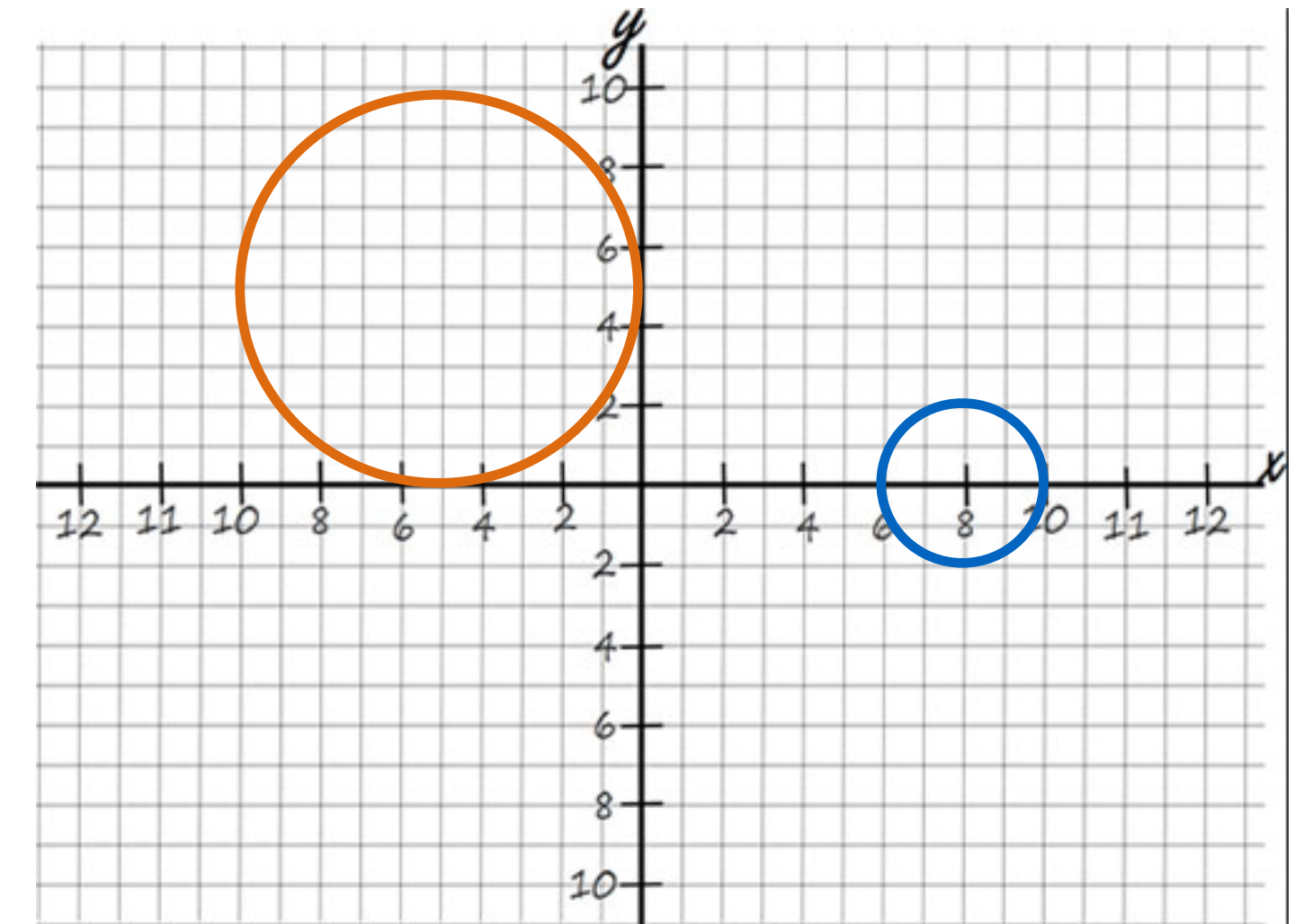
r = Rectangle(20, 10)
print(r.area())
```

Annotations and arrows point to specific parts of the code:

- Nombre de la clase (del tipo de dato)**: Points to the `Rectangle` name in the `class` statement.
- Constructor**: Points to the `def __init__` method.
- Atributos (variables)**: Points to the `self.w = width` and `self.h = height` lines.
- Método (función)**: Points to the `def area` method.
- Python es así. El primer parámetro representa la **instancia** de una clase.**: A pink arrow points from this text to the `self` parameter in the `__init__` method.
- A pink arrow points from the same text to the `Rectangle(20, 10)` instantiation.

Ejemplo: tipo de dato círculo

- **Objetivo:** crear un tipo de dato para manipular círculos en el plano.
- **Set de valores:** tres números reales
 - Posición en el plano y el radio.
- **Operaciones:**
 - Crear un nuevo círculo en posición (x,y) con radio r.
 - Determinar su distancia al origen.
 - Convertirlo a string



```
class Circle:
    """Creates a circle at (px, py) with radio r."""
```

Constructor

```
def __init__(self, px, py, r):
```

```
    self.x = px
    self.y = py
    self.radio = r
```

Atributos

Métodos

```
def dist(self):
    return sqrt(self.x*self.x + self.y*self.y)
```

```
def __str__(self):
    return "Circle at ({} , {}) with radio {}." \
           .format(self.x, self.y, self.radio)
```

```
c = Circle(10, 10, 5)
print(c.dist())
```

→ 14.142135623730951

```
help(Circle)
```

Help on class Circle in module __main__:

```
class Circle(builtins.object)
    Circle(px, py, r)
```

docstring,
documentación!

```
    Creates a circle at (px, py) with radio r.
```

Methods defined here:

```
    __init__(self, px, py, r)
        Initialize self. See help(type(self)) for accurate signatur
```

```
    __str__(self)
        Return str(self).
```

```
    dist(self)
```


Objeto: instancia de un tipo de dato

- Un objeto es una instancia de nuestro tipo de dato.
 - Si, es lo mismo que una variable (pero de nuestro tipo de dato!).

```
1 class Rectangle:
2     def __init__(self, width, height):
3         self.w = width
4         self.h = height
5
6     def area(self):
7         return self.w * self.h
8
9 def main():
10     rA = Rectangle(3, 4)
11     rB = Rectangle(9, 8)
12     print('area rA:', rA.area())
13     print('area rB:', rB.area())
14
15 if __name__ == '__main__': main()
```

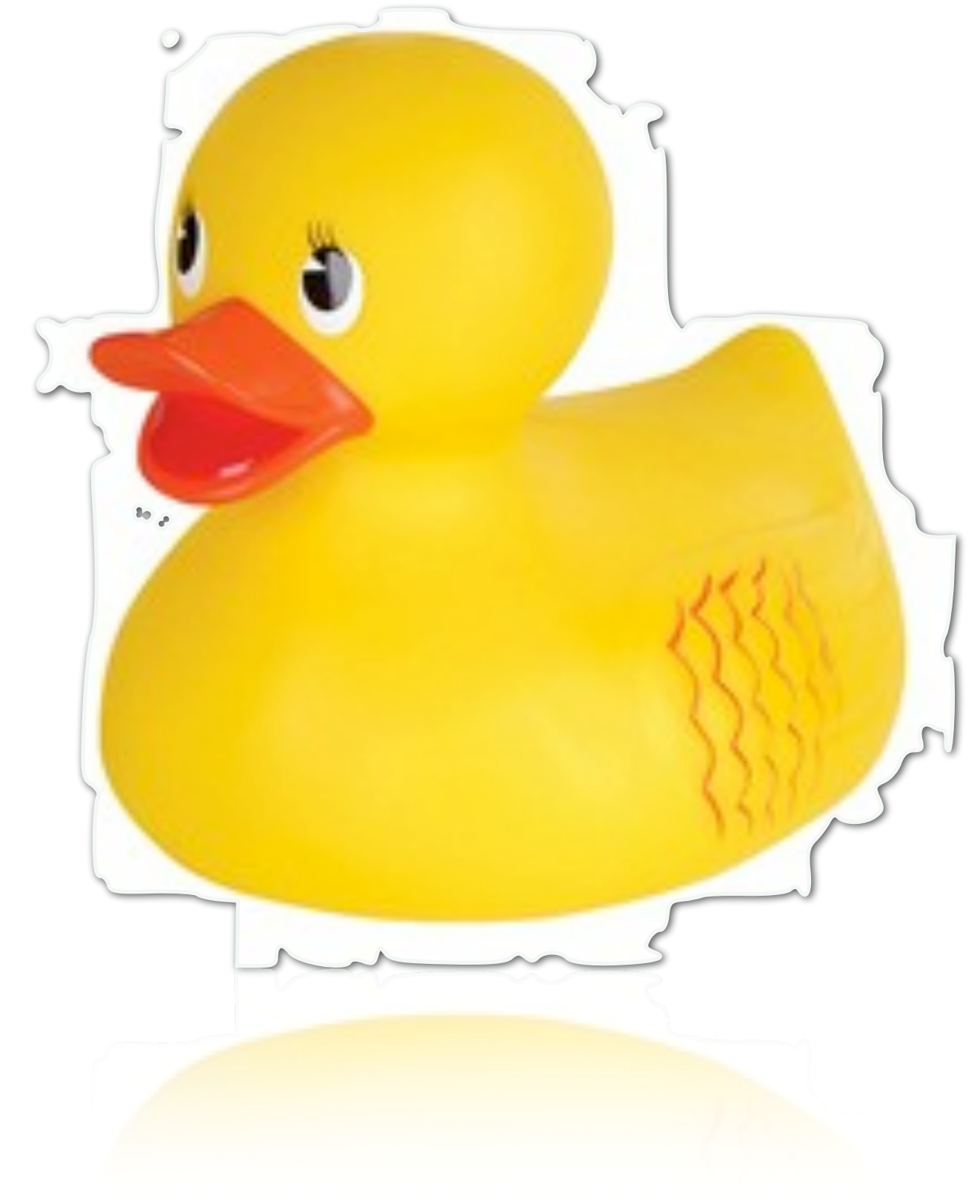
rA y rB son objetos de tipo Rectangle.

Método (función) ejecutado en objeto rB.

```

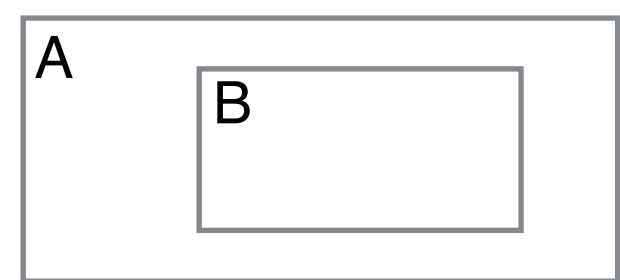
1 class Point:
2     """
3     def __init__(self, x, y):
4         self.x = x
5         self.y = y
6
7     def distX(self, p): return abs(self.x - p.x)
8     def distY(self, p): return abs(self.y - p.y)
9
10 class Rectangle:
11     """
12     """
13     def __init__(self, p1, p2):
14         self.p1 = p1
15         self.p2 = p2
16
17     def area(self):
18         return self.p1.distX(self.p2) * self.p1.distY(self.p2)
19
20 def main():
21     p1 = Point(-1, -1)
22     p2 = Point(1, 1)
23     r = Rectangle(p1, p2)
24     print('area de r:', r.area())
25
26 if __name__ == '__main__': main()

```



Método ejecutado
en objeto p1 que es parte de
Rectangle.

- **Objetivo:** implementar un tipo de dato para manejar rectángulos en el plano.
- **Conjunto de valores:** el punto de la esquina inferior-izquierda y el punto superior-derecha del rectángulo.
- **Operaciones:** devolver el área.
- Preguntas:
 - ¿Cómo podemos saber si dos rectángulos están solapados o están contenidos?
 - ¿Cómo podemos calcular el área de la intersección?



B está contenido en A



A y B están solapados. Área intersección.

```

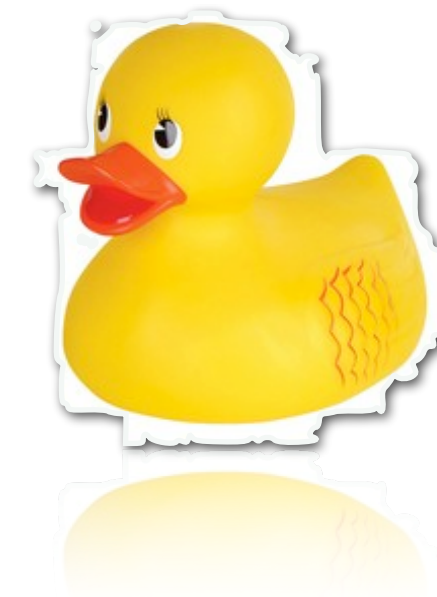
1 class Point:
2     """A 2d point in (x, y)."""
3     def __init__(self, x, y):
4         self.x = x
5         self.y = y
6
7     def distX(self, p): return abs(self.x - p.x)
8     def distY(self, p): return abs(self.y - p.y)
9
10 class Rectangle:
11     """A rectangle with the lower-left corner in
12     p1 and upper-right corner in p2."""
13     def __init__(self, p1, p2):
14         self.p1 = p1
15         self.p2 = p2
16
17     def area(self):
18         return self.p1.distX(self.p2) * self.p1.distY(self.p2)
19
20 def main():
21     p1 = Point(-1, -1)
22     p2 = Point(1, 1)
23     r = Rectangle(p1, p2)
24     print('area de r:', r.area())
25
26 if __name__ == '__main__': main()

```

Precondiciones

- ¿Cómo podemos asegurarnos que el usuario (final, u otra programadora) está usando correctamente nuestra API?

```
1 class Point:
2     """A 2d point in (x, y)."""
3     def __init__(self, x, y):
4         self.x = x
5         self.y = y
6
7     def __str__(self):
8         return 'is a point at ({}, {})'.format(self.x, self.y)
9
10 def main():
11     p1 = Point(-1, -1)
12     print('p1', p1)
13     p2 = Point('hola', 'chao')
14     print('p2', p2)
15
16 if __name__ == '__main__': main()
```



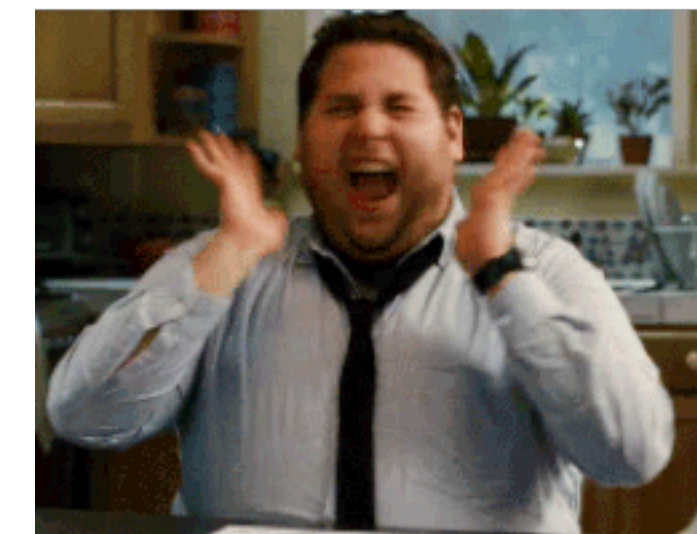
```
$ python3 point.py
p1 is a point at (-1, -1)
p2 is a point at (hola, chao)
```


Diseño por contrato

- **Precondición:** condición que se debe cumplir para ejecutar correctamente tu código.
- Condición debe estar definida utilizando **assert** condición, mensaje.
 - También podrías usar un **if**, pero con **assert** se termina la ejecución del programa si la condición es False.

```
1 class Point:
2     """A 2d point in (x, y)."""
3     def __init__(self, x, y):
4         assert isinstance(x, int), "x must be an integer"
5         assert isinstance(y, int), "y must be an integer"
6
7         self.x = x
8         self.y = y
9
10    def __str__(self):
11        return 'is a point at ({} , {})' .format(self.x, self.y)
12
13    def main():
14        p1 = Point(-1, -1)
15        print('p1', p1)
16        p2 = Point('hola', 'chao')
17        print('p2', p2)
18
19    if __name__ == '__main__': main()
```

```
$ python3 point-fixed.py
p1 is a point at (-1, -1)
Traceback (most recent call last):
  File "point-fixed.py", line 19, in <module>
    if __name__ == '__main__': main()
  File "point-fixed.py", line 16, in main
    p2 = Point('hola', 'chao')
  File "point-fixed.py", line 4, in __init__
    assert isinstance(x, int), "x must be an integer"
AssertionError: x must be an integer
```



Diseño por contrato

- **Postcondición:** condición que se debe cumplir una vez que se ejecuta tu código. Sirve para chequear que todo se ejecutó correctamente.

```
1 class Point:
2     """A 2d point in (x, y)."""
3     def __init__(self, x, y):
4         assert isinstance(x, int) or isinstance(x, float), 'x must be a number'
5         assert isinstance(y, int) or isinstance(y, float), 'y must be a number'
6         self.x = x
7         self.y = y
8
9     def distX(self, p): return abs(self.x - p.x)
10    def distY(self, p): return abs(self.y - p.y)
11
12 class Rectangle:
13     """A rectangle with the lower-left corner in
14     p1 and upper-right corner in p2."""
15     def __init__(self, p1, p2):
16         self.p1 = p1
17         self.p2 = p2
18
19     def area(self):
20         a = self.p1.distX(self.p2) * self.p1.distY(self.p2)
21         assert (a >= 0)
22         return result
```

Área debe ser no-negativa.

Ejercicio (en clases)

- **Objetivo:** implementar un tipo de dato para manejar números racionales (fracciones).
- **Conjunto de valores:** ... debe hacerlo usted...
- **Operaciones:**
 - suma, resta, multiplicación y división
 - devolver una representación en string

API:

class Rational

	Rational(n, m)	Crea un número racional de numerador n, y denominador m
Rational	plus(Rational b)	Suma de racional actual con b.
Rational	minus(Rational b)	Diferencia de racional actual con b.
Rational	times(Rational b)	Producto de racional actual con b.
Rational	divides(Rational b)	Cuociente de racional actual con b.
str	__str__()	Representación en string del número racional

Resumen

Conceptos:

- Clase: conjunto de valores y operaciones sobre esos valores.
- Atributos: conjunto de valores de una clase
- Métodos: funciones (u operaciones) de una clase
- Objetos: instancia de una clase (una variable del tipo de dato que creaste)

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

https://docs.python.org/3/reference/lexical_analysis.html

Funciones

- isinstance: chequea si una variable corresponde a un tipo de dato
- assert: detiene la ejecución del programa si una condición no se cumple

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

<https://docs.python.org/3/library/functions.html>