Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Clase 18: Programación Orientada a Objetos (II)

Rodrigo Toro Icarte (rntoro@uc.cl)

IIC1103 Introducción a la Programación - Sección 5

20 de Mayo, 2015

Clases pasadas

Hemos visto distintos tipos de datos en Python:

`int`, `float`, `complex`, `bool`, `string`, `list` y `tuple`.

Clases pasadas

Hemos visto distintos tipos de datos en Python:

`int`, `float`, `complex`, `bool`, `string`, `list` y `tuple`.

... y cómo crear nuestros propios tipos de datos:

Sintaxis

class nombre_clase**:**
    bloque_codigo_clase
(...)
variable = **nombre_clase()**

Clases pasadas

Cada clase tiene:

- Atributos → Variables
- Comportamiento → Métodos

Clases pasadas

Cada clase tiene:

- Atributos → Variables
- Comportamiento → Métodos

zombie
- vida - daño - velocidad - posición - dibujo
+ comer() + avanzar()

Clases pasadas

zombie
- vida - daño - velocidad - dibujo
+ comer() + avanzar()

Clases pasadas

Clase

```
class nombre_clase:
    def __init__(self, par_1, par_2, ...):
        self.Atributo_i = valor_i
        Resto_bloque_código_constructor
    def nombre_método(self, par_1, par_2, ...):
        Bloque_código_método
```

Objeto

```
var = nombre_clase()
var.Atributo_i = nuevo_valor
var.nombre_método(val_1, val_2, ...)
```

Clases pasadas

Clase:

```
2  class persona:
3    # Constructor
4    def __init__(self, nombre, apellido, n_alumno):
5      # Atributos de persona
6      self.nombre = nombre
7      self.apellido = apellido
8      self.n_alumno = n_alumno
9      self.notas = []
```

Objetos (instancias) de la clase:

```
12 juan = persona('Juan', 'Águila', '14000000')
13 aldo = persona('Aldo', 'Verri', '14000001')
14 maria = persona('María', 'Pinto', '14000002')
```

Tipos de datos

Existen 2 tipos de datos:

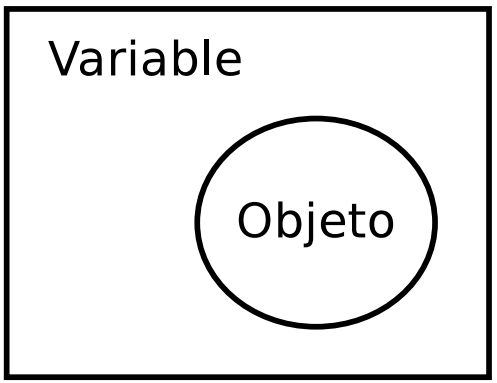
- Por valor.
- Por referencia.

Tipos de datos por valor

Por valor: La variable *contiene* al objeto.

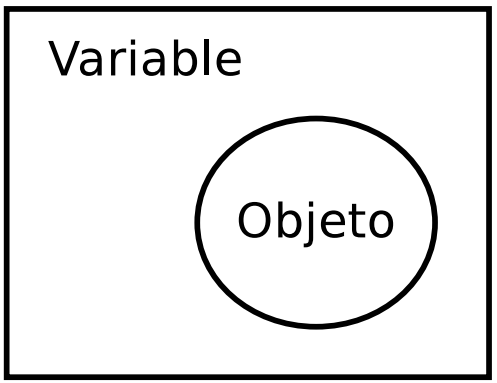
Tipos de datos por valor

Por valor: La variable *contiene* al objeto.



Tipos de datos por valor

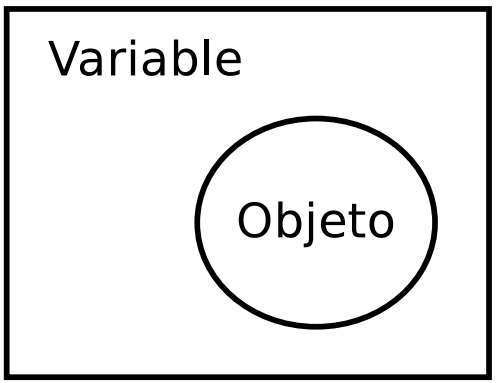
Por valor: La variable *contiene* al objeto.



Ejemplos: `int`, `float`, `complex` y `bool`.

Tipos de datos por valor

Por valor: La variable *contiene* al objeto.



Ejemplos: `int`, `float`, `complex` y `bool`.

Consecuencia: `b = a`, copia objeto `a` en `b`.

Tipos de datos por valor

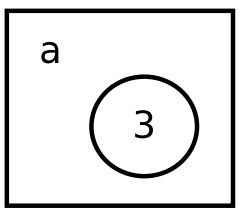
Ejemplo 1:

```
1 a = 3
2 b = a
3 b += 2
4 print("a =",a,", b =",b)
```

Tipos de datos por valor

Ejemplo 1:

```
1 a = 3
2 b = a
3 b += 2
4 print("a =",a,", b =",b)
```

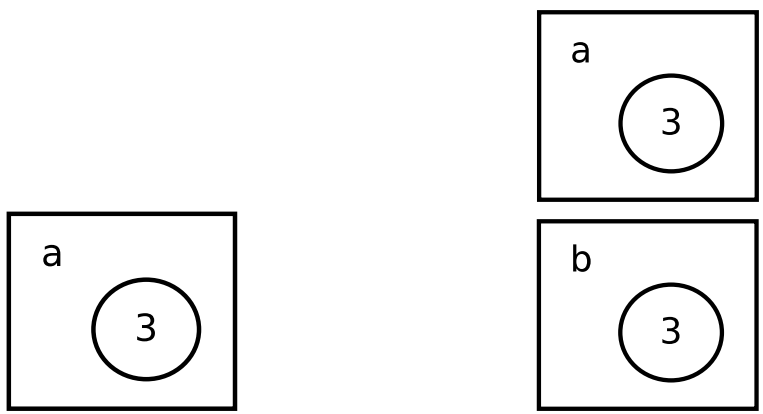


(a) a = 3

Tipos de datos por valor

Ejemplo 1:

```
1 a = 3
2 b = a
3 b += 2
4 print("a =",a,", b =",b)
```

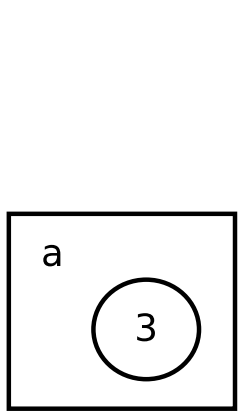


(a) `a = 3` (b) `b = a`

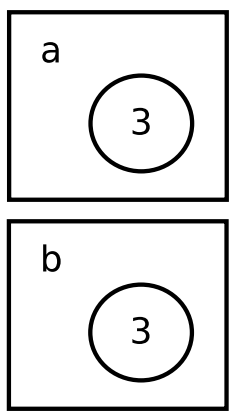
Tipos de datos por valor

Ejemplo 1:

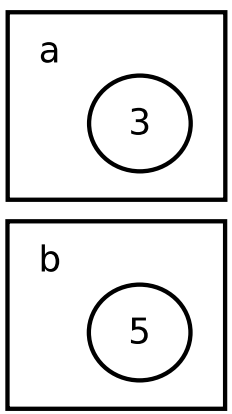
```
1 a = 3
2 b = a
3 b += 2
4 print("a =",a,", b =",b)
```



(a) `a = 3`



(b) `b = a`



(c) `b += 2`

Tipos de datos por valor

Ejemplo 2:

```
1 def f(b):
2   b += 2
3
4 a = 3
5 f(a)
6 print(a)
```

¿Qué imprime `print(a)`?

Tipos de datos por valor

Ejemplo 2:

```
1 def f(b):
2   b += 2
3
4 a = 3
5 f(a)
6 print(a)
```

¿Qué imprime `print(a)`?

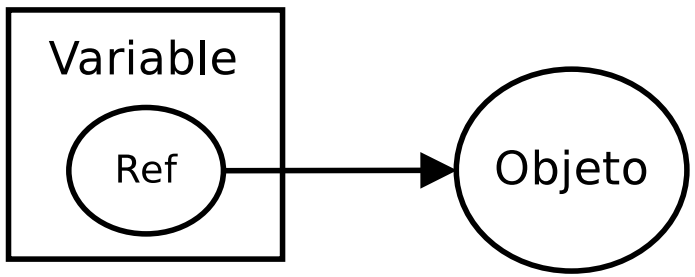
Observación: Al dar `a` como parámetro implícitamente se hace un `b = a`.

Tipos de datos por referencia

Por referencia: La variable *contiene* una referencia al objeto.

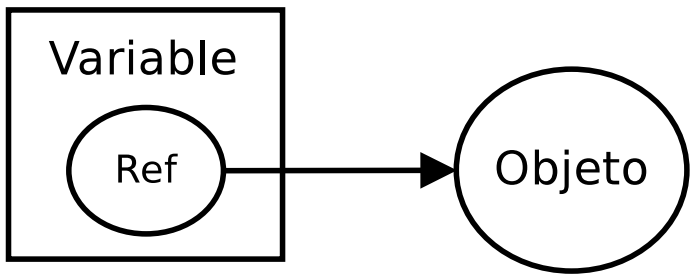
Tipos de datos por referencia

Por referencia: La variable *contiene* una referencia al objeto.



Tipos de datos por referencia

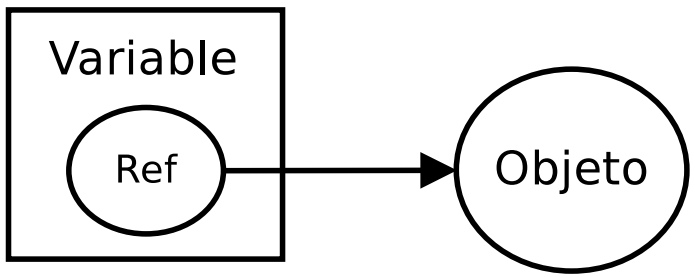
Por referencia: La variable *contiene* una referencia al objeto.



Ejemplos: `string`, `tuple`, `list` y clases creadas por nosotros.

Tipos de datos por referencia

Por referencia: La variable *contiene* una referencia al objeto.



Ejemplos: `string`, `tuple`, `list` y clases creadas por nosotros.

Consecuencia: `b = a`, copia la referencia al objeto `a` en `b`.

Tipos de datos por referencia

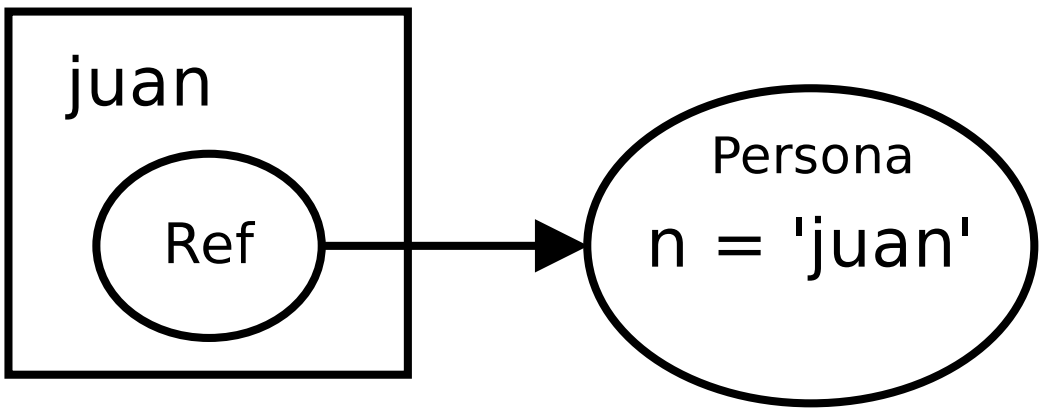
Ejemplo 1:

```
1 class persona:
2   def __init__(self, n):
3     self.n = n
4
5 juan = persona('juan')
6 pedro = juan
7 pedro.n = 'pedro'
8 print(juan.n)
```

¿Qué imprime `print(juan.n)`?

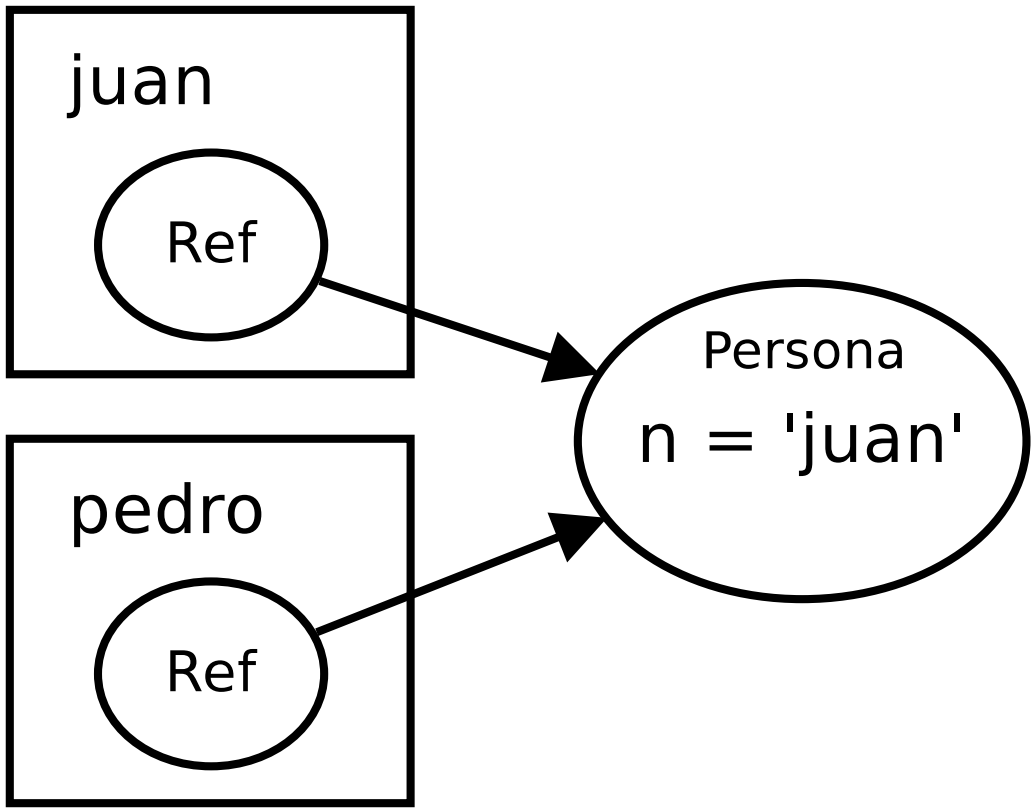
Tipos de datos por referencia

```
juan = persona('juan')
```



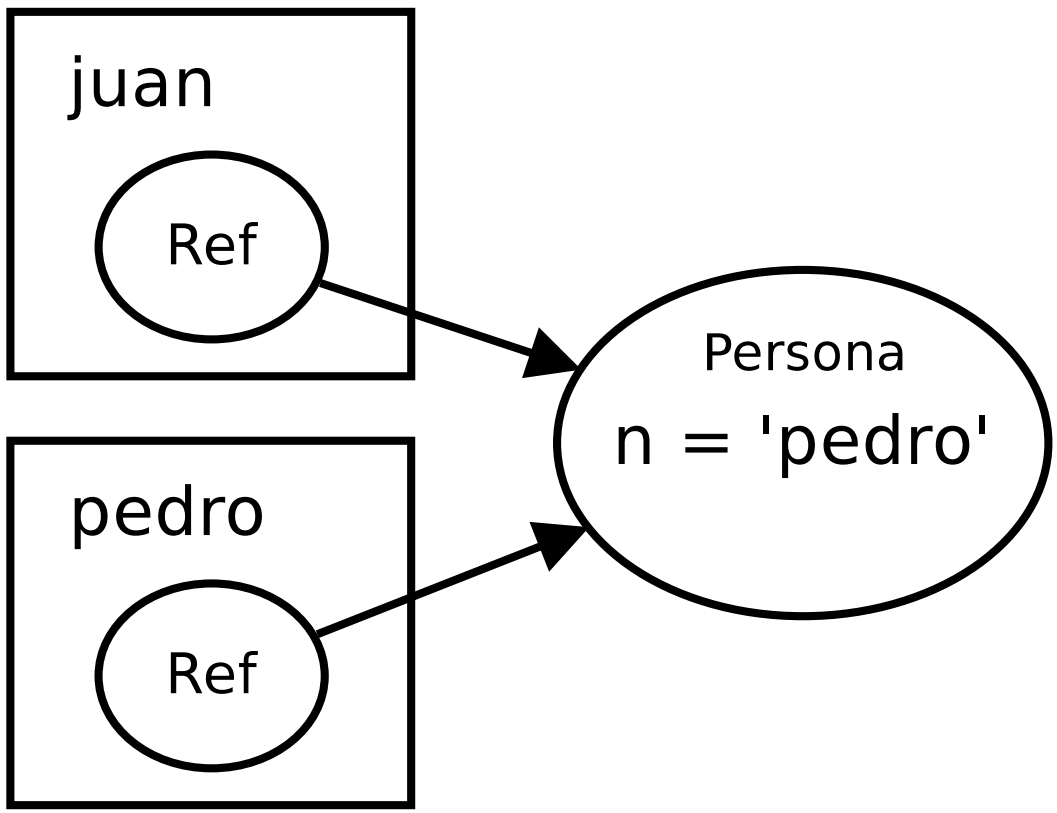
Tipos de datos por referencia

```
pedro = juan
```



Tipos de datos por referencia

```
pedro.n = 'pedro'
```



Tipos de datos por referencia

Ejemplo 2:

```
1  class persona:
2    def __init__(self, n):
3      self.n = n
4
5  def f(b):
6    b.n = 'pedro'
7
8  juan = persona('juan')
9  f(juan)
10 print(juan.n)
```

¿Qué imprime `print(juan.n)`?

Tipos de datos por referencia

Ejemplo 2:

```
1  class persona:
2    def __init__(self, n):
3      self.n = n
4
5  def f(b):
6    b.n = 'pedro'
7
8  juan = persona('juan')
9  f(juan)
10 print(juan.n)
```

¿Qué imprime `print(juan.n)`?

Observación: Al dar `juan` como parámetro implícitamente se hace un `b = juan`.

Tipos de datos por referencia

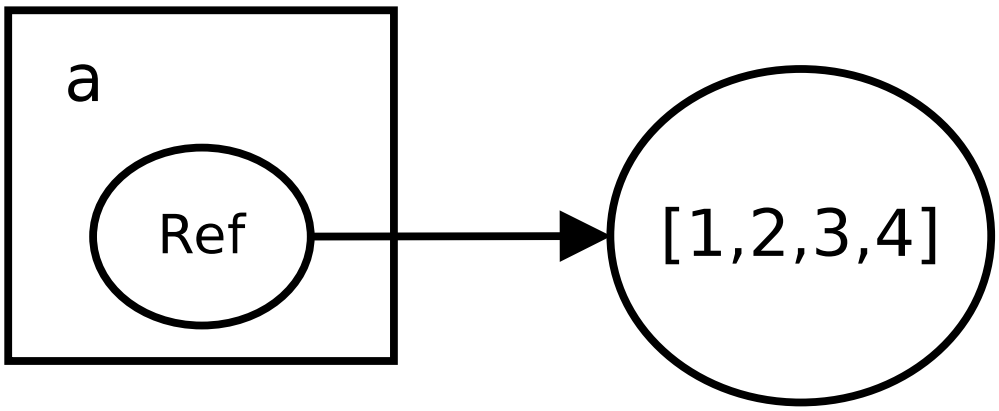
Ejemplo 3:

```
1  def quitar_minimo(l1):
2    l1.remove(min(l1))
3
4  def vaciar_lista(l2):
5    l2 = []
6
7  a = [1,2,3,4]
8  quitar_minimo(a)
9  print(a)
10 vaciar_lista(a)
11 print(a)
```

¿Qué ocurre en este caso?

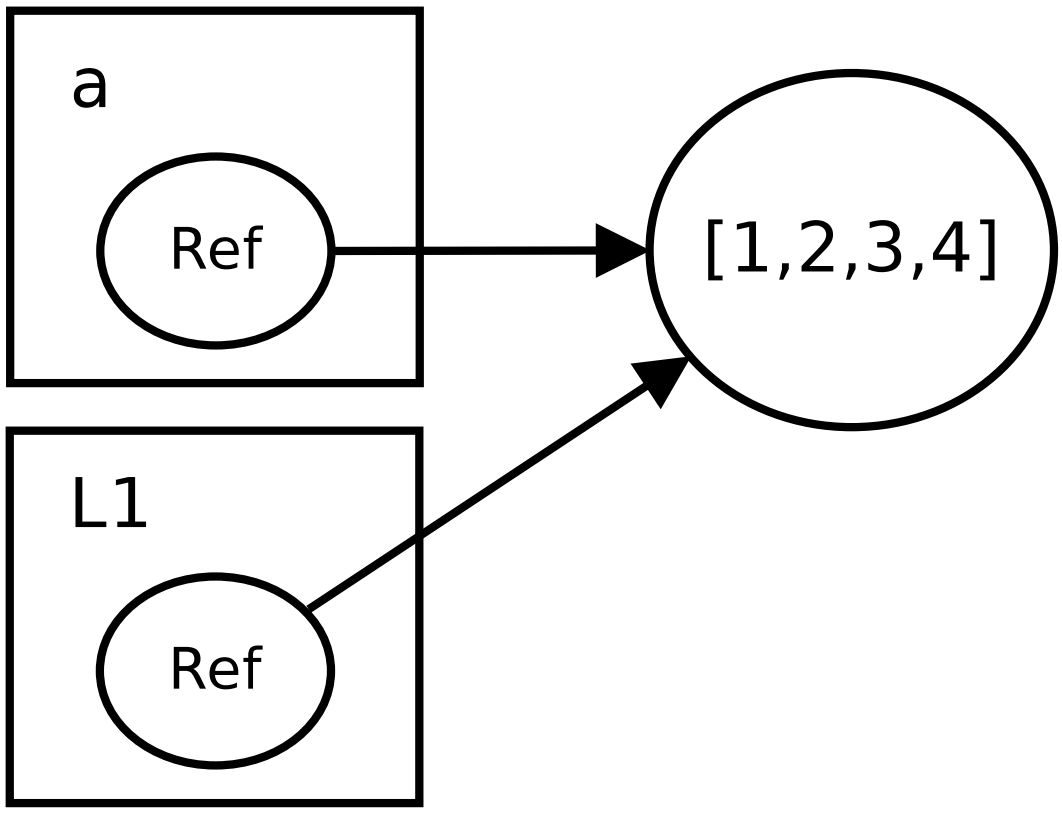
Tipos de datos por referencia

```
a = [1,2,3,4]
```



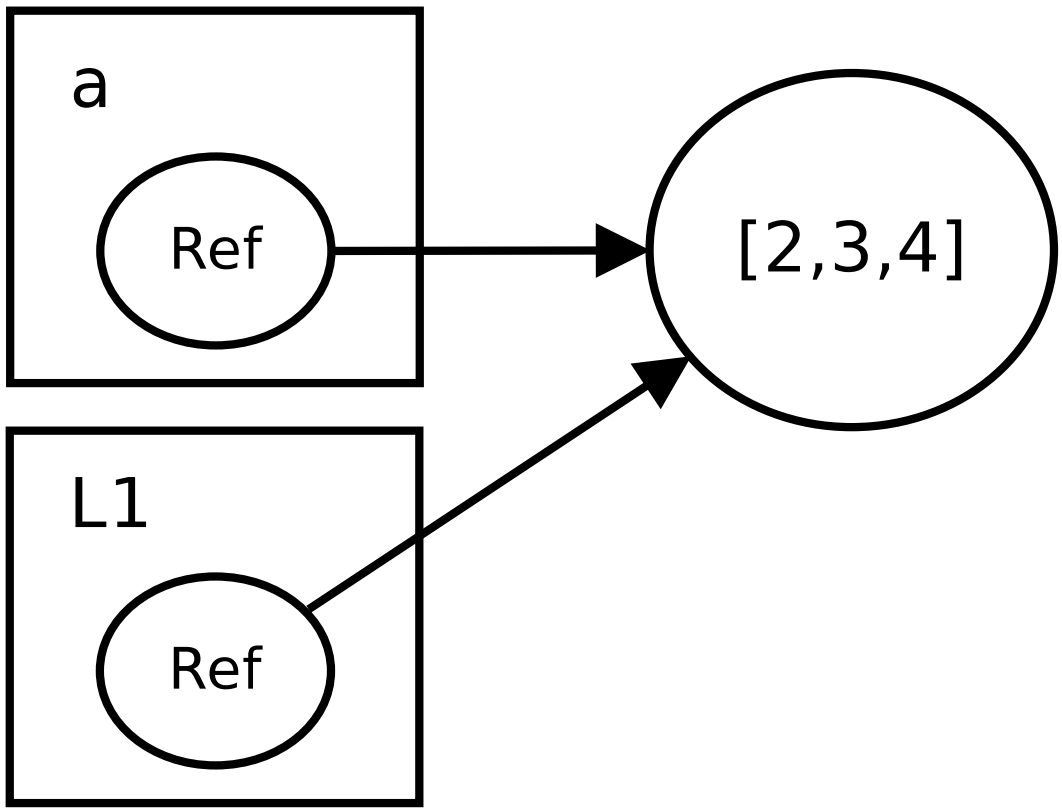
Tipos de datos por referencia

`quitar_minimo(a)`



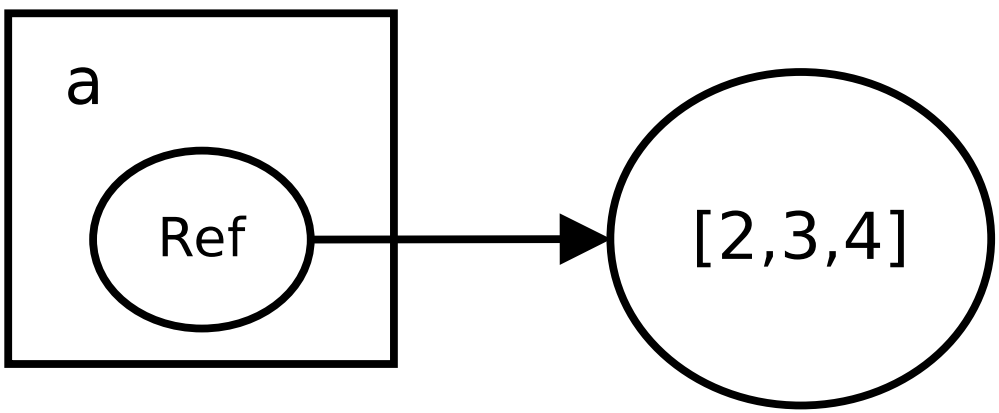
Tipos de datos por referencia

```
l1.remove(min(l1))
```

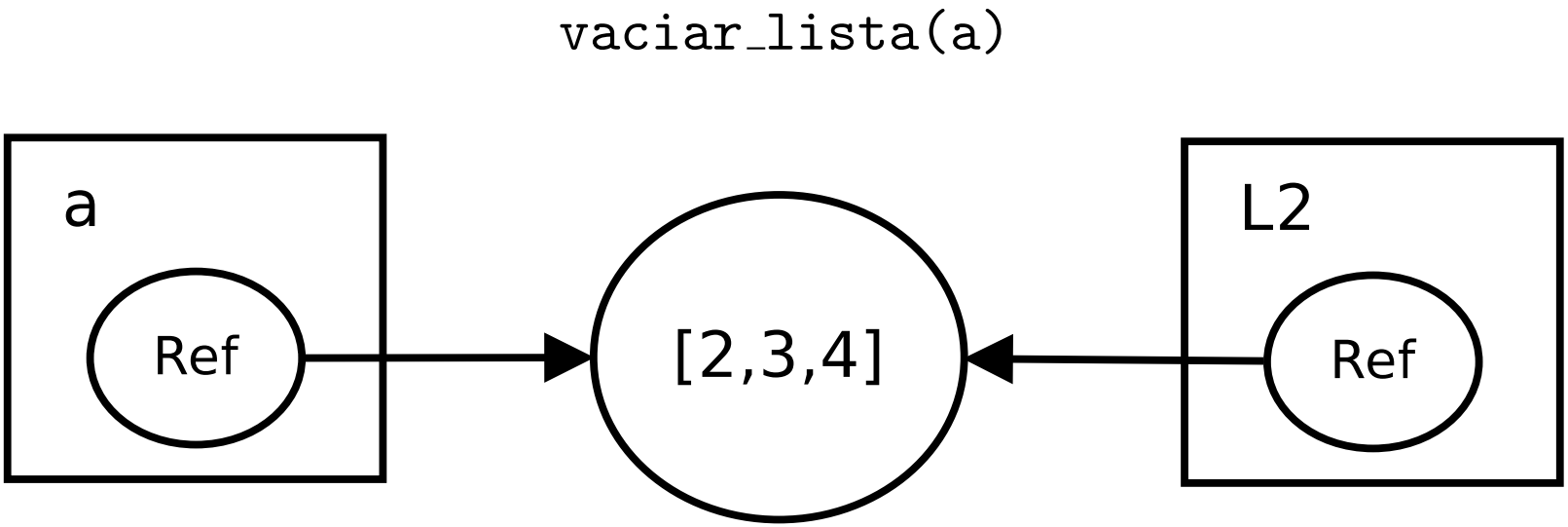


Tipos de datos por referencia

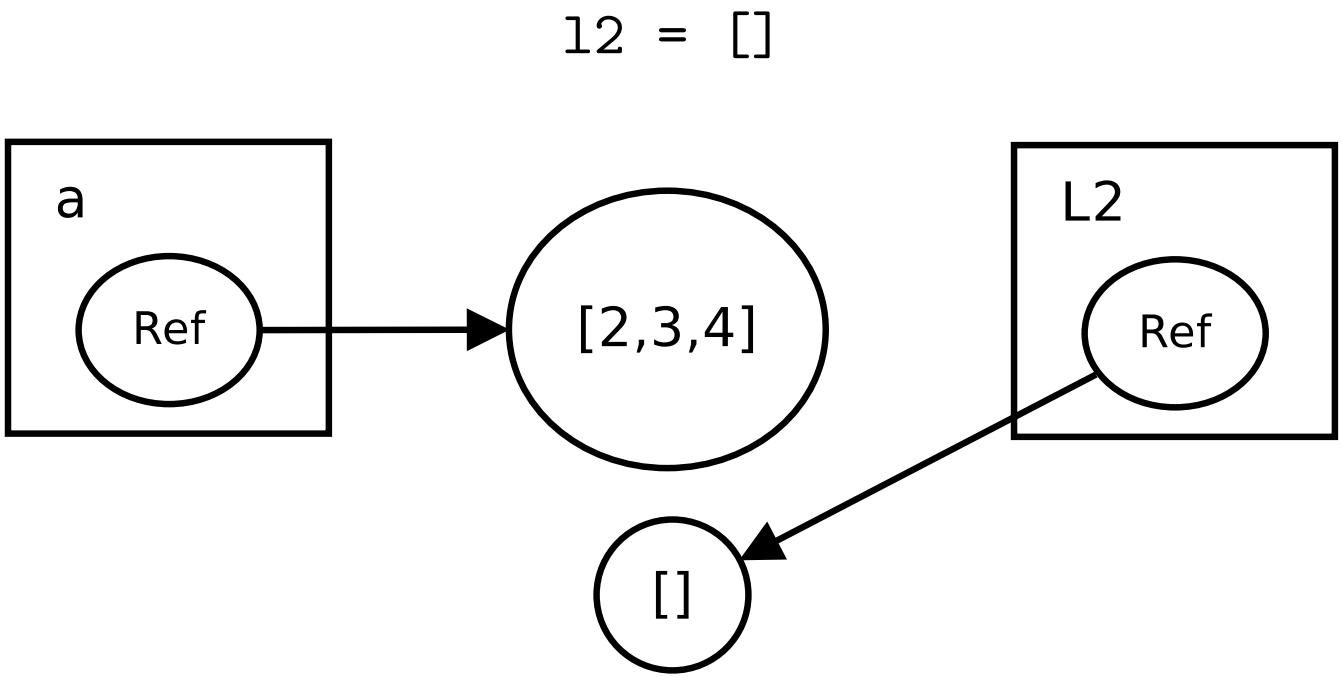
```
print(a)
```



Tipos de datos por referencia

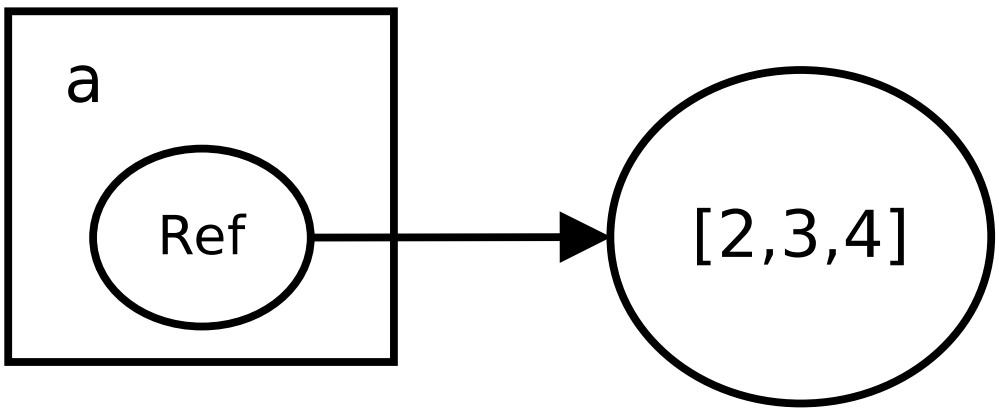


Tipos de datos por referencia



Tipos de datos por referencia

```
print(l)
```



Tipos de datos por referencia

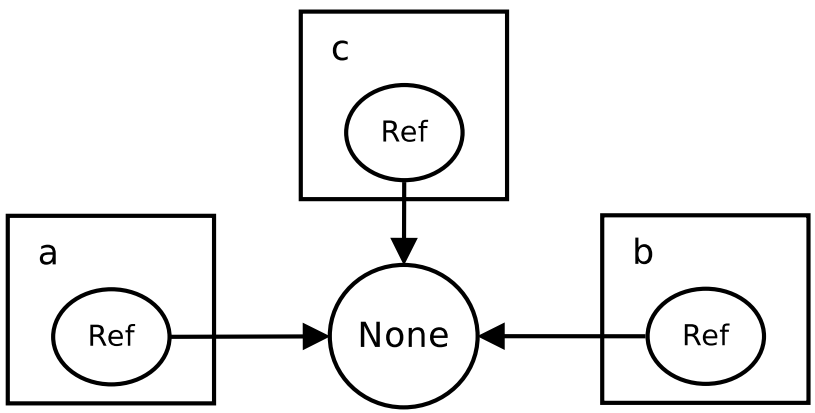
Ejemplo 4:

```
1  class persona:
2    def __init__(self, nombre):
3      self.nombre = nombre
4      self.notas = []
5    def agregar_notas(self,l):
6      self.notas.extend(l)
7
8  juan = persona('Juan')
9  n = juan.notas
10 juan.agregar_notas([6.5, 7.0, 6.7])
11 del n[:]
12 print(juan.notas)
```

None

Tipo especial de dato que apuntan al mismo objeto `NoneType`.

```
1 a = None
2 b = None
3 c = None
```



None

Con `None` no se puede hacer nada:

None

Con `None` no se puede hacer nada:

- No tiene atributos.
- No tiene métodos.
- No le puedes agregar atributos.
- (...)

None

Con `None` no se puede hacer nada:

- No tiene atributos.
- No tiene métodos.
- No le puedes agregar atributos.
- (...)

¿Para qué sirve? :s

None

Con `None` no se puede hacer nada:

- No tiene atributos.
- No tiene métodos.
- No le puedes agregar atributos.
- (...)

¿Para qué sirve? :s

... para usarlo como valor inválido.

None

Ej: Implementa la función `obtener_persona(l,nombre)`, que recibe una lista de objetos del tipo *personas* (que tiene como atributo su `nombre`, entre otras cosas), y retorna la persona llamada `nombre`.

None

Ej: Implementa la función `obtener_persona(l,nombre)`, que recibe una lista de objetos del tipo *personas* (que tiene como atributo su `nombre`, entre otras cosas), y retorna la persona llamada `nombre`.

```
6 def obtener_persona(l,nombre):
7   for p in l:
8     if(p.nombre == nombre):
9       return p
```

None

Ej: Implementa la función `obtener_persona(l,nombre)`, que recibe una lista de objetos del tipo *personas* (que tiene como atributo su `nombre`, entre otras cosas), y retorna la persona llamada `nombre`.

```
6 def obtener_persona(l,nombre):
7   for p in l:
8     if(p.nombre == nombre):
9       return p
```

¿Qué debiera retornar si `nombre` no se encuentra en `l`?

None

```
6  def obtener_persona(l,nombre):
7    for p in l:
8      if(p.nombre == nombre):
9        return p
10   return None
```

None

```
6  def obtener_persona(l,nombre):
7    for p in l:
8      if(p.nombre == nombre):
9        return p
10   return None
```

... al llamar a la función se chequea si el resultado es `None`.

```
15 p = obtener_persona(l,"pedro")
16 if(p is None):
17   print("persona no encontrada")
```

None

```
1  class persona:
2    def __init__(self,nombre,apellido):
3      self.nombre = nombre
4      self.apellido = apellido
5
6  def obtener_persona(l,nombre):
7    for p in l:
8      if(p.nombre == nombre):
9        return p
10   return None
11
12 l = [persona("juan","águila"),
13       persona("maría","pinto"),
14       persona("aldo","verri")]
15 p = obtener_persona(l,"pedro")
16 if(p is None):
17   print("persona no encontrada")
18 else:
19   print("encontré a",p.nombre,p.apellido)
```

Tipos de datos

¿Por qué existen distintos tipos de datos?

Relaciones entre clases

Las clases interactúan entre ellas.

Relaciones entre clases

Existen 3 tipos de relaciones entre clases, pero veremos 2:

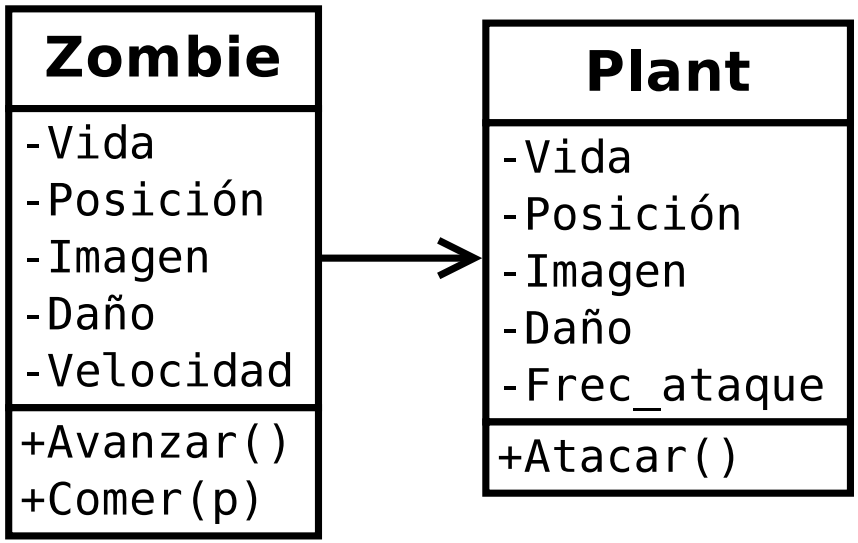
- Asociación.
- Composición.

Relaciones entre clases

Asociación: Se produce si una clase utiliza a la otra en alguno de sus métodos.

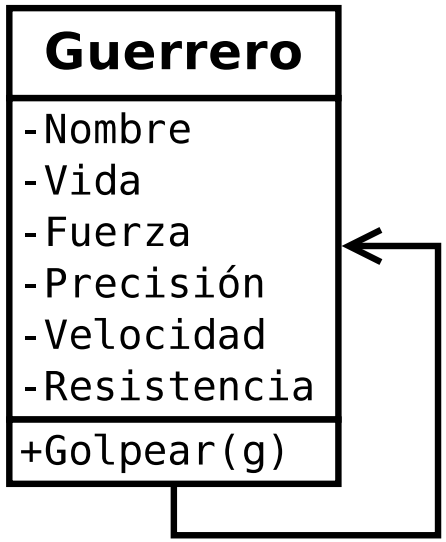
Relaciones entre clases

Asociación: Se produce si una clase utiliza a la otra en alguno de sus métodos.



Relaciones entre clases

Asociación: Se produce si una clase utiliza a la otra en alguno de sus métodos.

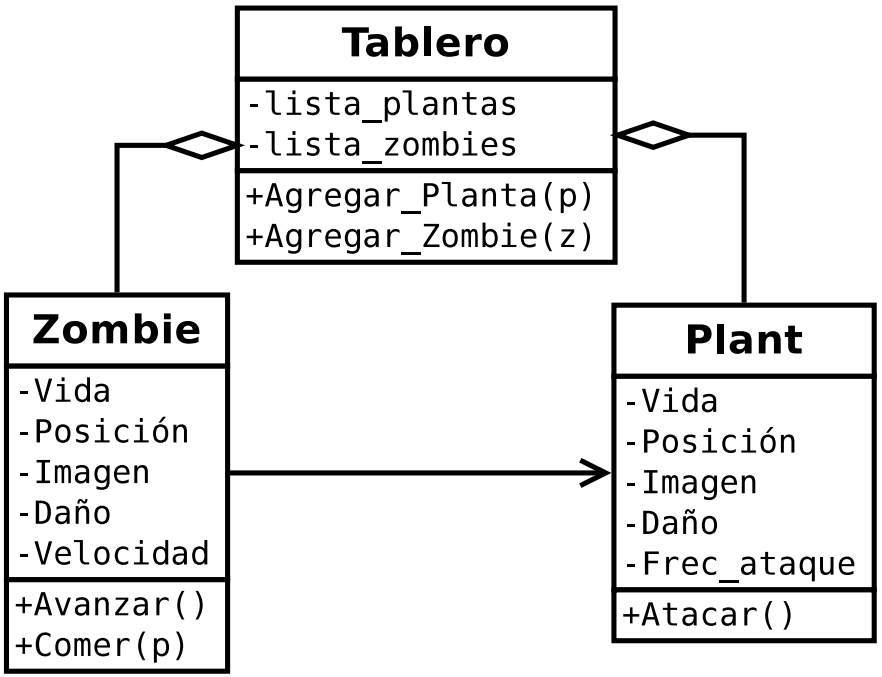


Relaciones entre clases

Composición: Se produce si una clase tiene como atributo a otra clase.

Relaciones entre clases

Composición: Se produce si una clase tiene como atributo a otra clase.



Ejemplo: Blackjack

"Cree un programa que permita jugar Blackjack contra el computador"

Ejemplo: Blackjack

Reglas:

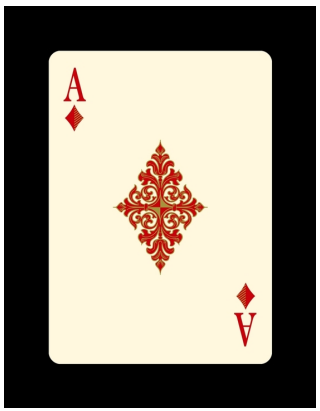
- Se juega con un mazo de cartas inglesas.
- Se entregan 2 cartas al jugador y 2 a la máquina.
- El computador puede ver ambas cartas del jugador, pero el jugador solo ve la primera carta del computador.
- Si la suma de la mano inicial del jugador es 21, gana automáticamente (donde **J**, **Q** y **K** valen 10 y el **As** puede valer 11 o 1).
- En otro caso, el jugador puede pedir cartas mientras no supere 21.
- Cuando el jugador decida no pedir más cartas, el computador comenzará a pedir hasta que supere la suma del jugador o se pase de 21.
- Gana quien esté más cercano a 21 por debajo (pueden existir empates).

Ejemplo: Blackjack

¿Qué clases deberíamos considerar?

Ejemplo: Blackjack

¿Qué clases deberíamos considerar?

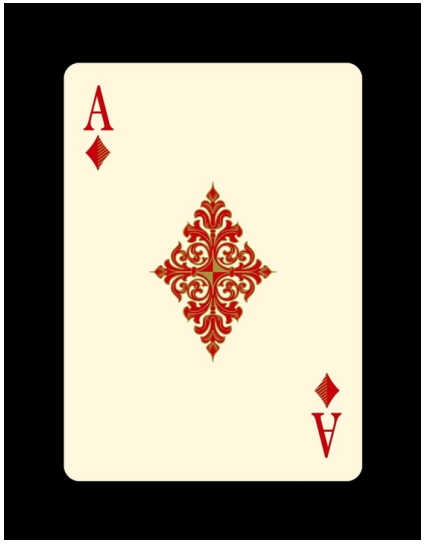


(a) Carta

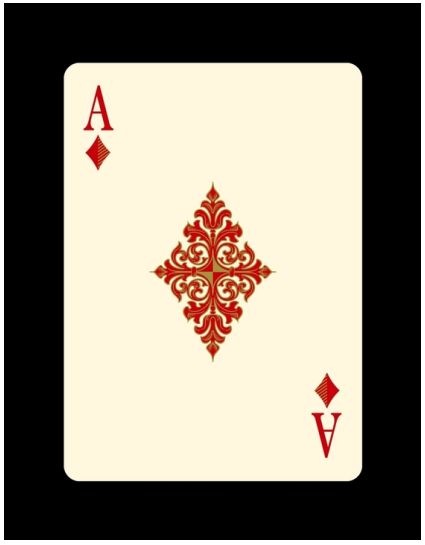
(b) Mazo

(c) Jugador

Ejemplo: Blackjack



Ejemplo: Blackjack



Carta
- número - pinta
+ get_valor()

Ejemplo: Blackjack

Ejemplo: Blackjack

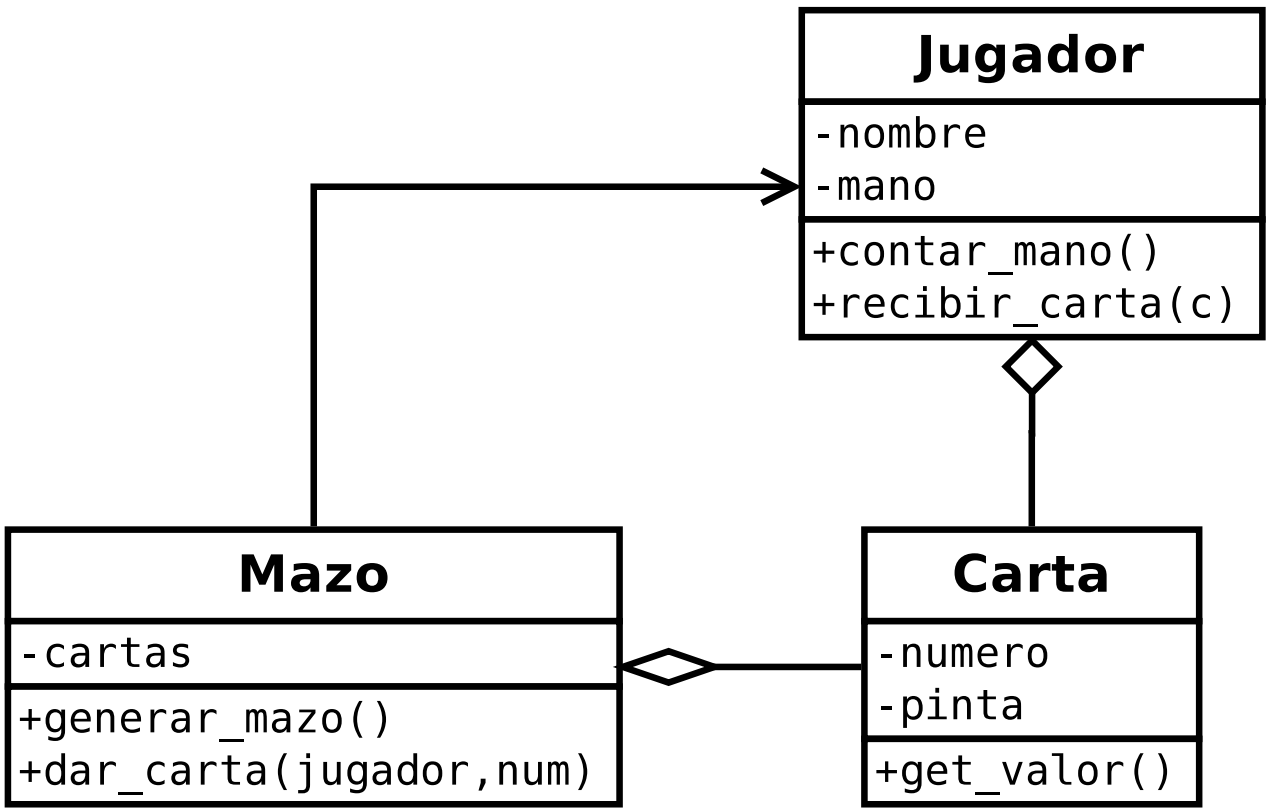
Mazo
- cartas
+ generar_mazo() + dar_carta(jugador, num)

Ejemplo: Blackjack

Ejemplo: Blackjack

Jugador
- nombre - mano
+ contar_mano() + recibir_carta(c)

Ejemplo: Blackjack



Sobrecarga de métodos

Python permite sobrecargar métodos en la definición de la clase.

... es decir, definir comportamientos para +, *, <=, etc.

Sobrecarga de métodos

Python permite sobrecargar métodos en la definición de la clase.

... es decir, definir comportamientos para +, *, <=, etc.

```
1  class persona:
2    pass
3
4  a = persona()
5  b = persona()
6  print(str(a))    # ?
7  print(int(a))    # ?
8  print(a+b)       # ?
9  print(a*b)       # ?
10 print(a < b)     # ?
11 print(a and b)   # ?
```

Sobrecarga de métodos

Intentemos mejorar el *mini-siding*.

Sobrecarga de métodos

Intentemos mejorar el *mini-siding*.

```
2  class persona:
3
4    # Constructor
5    def __init__(self, nombre, apellido, n_alumno):
6      # Atributos de persona
7      self.nombre = nombre
8      self.apellido = apellido
9      self.n_alumno = n_alumno
10     self.notas = []
11
12   # Métodos
13   def agregar_nota(self,n):
14     self.notas.append(n)
15   def agregar_notas(self,l):
16     self.notas.extend(l)
17   def get_promedio(self):
18     return sum(self.notas)/len(self.notas)
```

Sobrecarga de métodos

```
20 # Creo las personas y doy valores a sus atributos
21 juan = persona('Juan', 'Águila', '14000000')
22 aldo = persona('Aldo', 'Verri', '14000001')
23 maria = persona('María', 'Pinto', '14000002')
24
25 # Agrego notas
26 juan.agregar_notas([6.5, 7.0, 6.7])
27 aldo.agregar_notas([3.0, 2.7, 3.8])
28 maria.agregar_notas([5.7, 7.0, 6.2])
29
30 # Formo lista y muestro promedios
31 estudiantes = [juan, aldo, maria]
32 for e in estudiantes:
33   print(e.apellido,"\t=>",'%0.2f'%e.get_promedio())
34
35 # Salida:
36 #    >>> Águila   => 6.73
37 #    >>> Verri    => 3.17
38 #    >>> Pinto    => 6.30
```

Sobrecarga de métodos

Sobrecarga de *casteadores*.

Nombre función	Descripción
`__int__(self)`	A int
`__float__(self)`	A float
`__bool__(self)`	A bool
`__str__(self)`	A string

Sobrecarga de métodos

Sobrecarga de *casteadores*.

Nombre función	Descripción
`__int__(self)`	A int
`__float__(self)`	A float
`__bool__(self)`	A bool
`__str__(self)`	A string

Para el *mini-siding* nos podría servir el `__str__(self)`.

Sobrecarga de métodos

Idea: Cuando muestro los datos de un alumno siempre uso el formato `Apellido promedio`. Definamos esto en `__str__(self)`.

```
21    def __str__(self):
22      s = self.apellido + "\t=> %0.2f"%e.get_promedio()
23      return s
```

Sobrecarga de métodos

Idea: Cuando muestro los datos de un alumno siempre uso el formato `Apellido promedio`. Definamos esto en `__str__(self)`.

```
21   def __str__(self):
22     s = self.apellido + "\t=> %0.2f"%e.get_promedio()
23     return s
```

Antes:

```
32 for e in estudiantes:
33   print(e.apellido,"\t=>",'%0.2f'%e.get_promedio())
```

Ahora:

```
37 for e in estudiantes:
38   print(e)
```

Sobrecarga de métodos

Sobrecarga de comparadores.

Nombre función	Descripción
`__lt__(self, other)`	Menor que
`__le__(self, other)`	Menor o igual
`__eq__(self, other)`	Igual
`__ne__(self, other)`	No es igual
`__gt__(self, other)`	Mayor que
`__ge__(self, other)`	Mayor o igual

Sobrecarga de métodos

Sobrecarga de comparadores.

Nombre función	Descripción
`__lt__(self, other)`	Menor que
`__le__(self, other)`	Menor o igual
`__eq__(self, other)`	Igual
`__ne__(self, other)`	No es igual
`__gt__(self, other)`	Mayor que
`__ge__(self, other)`	Mayor o igual

Para el *mini-siding* nos podría servir el `__gt__(self,other)`.

Sobrecarga de métodos

¿Cómo ordenamos por nota, rompiendo empates por apellido?

Sobrecarga de métodos

¿Cómo ordenamos por nota, rompiendo empates por apellido?

Idea: Si definimos `__gt__(self,other)` podríamos usar `sort()`.

```
28    def __gt__(self,other):
29      # Comparo según notas
30      if(self.get_promedio() > other.get_promedio()):
31        return True
32      elif(self.get_promedio() < other.get_promedio()):
33        return False
34      else: # si tienen la misma nota
35        if(self.apellido > other.apellido): return True
36        else: return False
```

Sobrecarga de métodos

¿Cómo ordenamos por nota, rompiendo empates por apellido?

Idea: Si definimos `__gt__(self,other)` podríamos usar `sort()`.

```
28   def __gt__(self,other):
29     # Comparo según notas
30     if(self.get_promedio() > other.get_promedio()):
31       return True
32     elif(self.get_promedio() < other.get_promedio()):
33       return False
34     else: # si tienen la misma nota
35       if(self.apellido > other.apellido): return True
36       else: return False
```

Ordenar es simplemente:

```
50 estudiantes.sort()
```

Sobrecarga de métodos

Sobrecarga de operadores matemáticos y lógicos.

Nombre función	Descripción
`__add__(self,other)`	a + b
`__sub__(self,other)`	a - b
`__mul__(self,other)`	a * b
`__div__(self,other)`	a / b
`__not__(self)`	not a
`__and__(self,other)`	a and b
`__or__(self,other)`	a or b

Sobrecarga de métodos

Sobrecarga de operadores matemáticos y lógicos.

Nombre función	Descripción
`__add__(self,other)`	a + b
`__sub__(self,other)`	a - b
`__mul__(self,other)`	a * b
`__div__(self,other)`	a / b
`__not__(self)`	not a
`__and__(self,other)`	a and b
`__or__(self,other)`	a or b

¿Alguna idea sobre qué podríamos hacer con `__add__(self,other)`?

Sobrecarga de métodos

Programe un simulador de batallas entre:

(a) Súperman.

(b) Gokú.

(c) Chuck Norris.

Sobrecarga de métodos

Usaremos una sola clase:

guerrero
- nombre - vida - fuerza - precisión - velocidad - defensa
+ golpear(g)

Sobrecarga de métodos

Sobreescribamos `__add__(self,other)` en la clase guerrero.

```
3  class guerrero:
4    def __add__(self,other):
5      nombre = self.nombre + "_" + other.nombre
6      vida = self.vida + other.vida
7      fuerza = self.fuerza + other.fuerza
8      precision = self.precision + other.precision
9      velocidad = self.velocidad + other.velocidad
10     defensa = self.defensa + other.defensa
11     return guerrero(nombre,vida,fuerza,precision,
12                     velocidad,defensa)
13   # ... más abajo "constructor" y "golpear"
```

Sobrecarga de métodos

Ahora podemos crear un súper guerrero...

```
48 # creamos guerreros
49 superman = guerrero('Superman',100,50,80,30,20)
50 goku = guerrero('Gokú',100,60,80,40,20)
51 chuck = guerrero('Chuck Norris',200,99,99,99,99)
52 # Creamos el super guerrero
53 supergoku = superman + goku
```

Sobrecarga de métodos

Ahora podemos crear un súper guerrero...

```
48 # creamos guerreros
49 superman = guerrero('Superman',100,50,80,30,20)
50 goku = guerrero('Gokú',100,60,80,40,20)
51 chuck = guerrero('Chuck Norris',200,99,99,99,99)
52 # Creamos el super guerrero
53 supergoku = superman + goku
```

... y simular la batalla

```
55 # simula batalla
56 simular_batalla(supergoku, chuck)
```

Ejemplo: Blackjack

¿Programemos esto?

Demo