Objects

Java

private package

fields

inheritance

design patterns

records

abstract class base public virtual

this/self

Factory

Singleton

interfaces

def constructors

protected

message passing

polymorphism

methods

OOP

extend

templates/generics

type classes

friend

composition

names → stuff (values, methods)

BLUE = OBJECT LANG

PINK = DESUGARED OUTPUT

Object ≡ StrngDict ①

StrngDict<Box> ②

③ Box<StrngDict>

o.fld

① o.get("fld")

if field not found ST

- error (Python, Ruby)
- undef (JS)

② get-box (o.get("fld"))

o.fld = v

if fld not found

- Python (adds field)
- Ruby, ST (throws error)

① o.set("fld", v)

record extension

② set-box(o.get("fld"), v)

③ set-box(o, get-box(o).set("fld" v))

{fld₁ : v₁, fld₂ : v₂, ...}

① [strng-dict: "fld₁", v₁ ...]

② [strng-dict: "fld₁", box(v₁) ...]

③ box([strng-dict: "fld₁", v₁, ...])

o.m(v1, ...)

o.get("m")(v1, ...)

```
o = { x: 4, y: 5,
    dist0: lam():
        sqrt(x² + y²)
    end
}
o.dist0()
```

```
letrec this: [sd: "x", 4, "y", 5,
    dist0: lam():
        sqrt((x * x) + (y * y))

        sqrt((this.x * this.x) +
             (this.y * this.y))
    end
]
this
```

Ruby — self

```
dist0: lam(self):
    sqrt(self.x * self.x
         self.y * self.y)
end

o:dist0()
o.dist0()
```

o.get("dist0")(o)

Python
Lua
JS

self passed @ call site

```
p = Point(4, 5)
fun Point(x, y):
    {
      dist0: lam():
          sqrt(x^2 + y^2)
      end
    }
end
fun f():
    p.x
end
```

simple private fields - just close over ids

(use var x, var y to get mutable fields)

```
p = Point(4, 5)

fun Point(x, y):
    { private x: x , private y: y,
      dist-to: lam(other):
          sqrt(this.x - other.x
               this.y - other.y )
      end
    }
end
fun f():
  p.x
end
```

More complicated private fields — desugar "class body" differently than rest of program

```
public class C {
    int i;
    void setI(int newI) {
      this.i = newI;
```