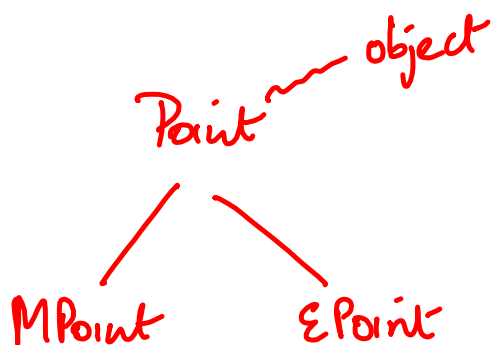


Inheritance ...

old type class vs New type class



A diagram illustrating inheritance for a new type class. It shows the code `super(MPoint, self)` in purple, with a green arrow pointing from `self` to `super`. Below it, `self.super()` is written in green, with a red arrow pointing from `self` to `super`. A red arrow points from the text "Which parent?" in red to the `super` in `self.super()`.

XYPoint

x, y

getpoint()

setpoint()

PolarPoint

r, theta

getpoint()

setpoint()

MPoint (x, y, t)

distance()

{
"xy"
"polar"}

Multiple Inheritance

```
class MPoint (XYPoint, PolarPoint):
```

```
    def __init__ (self, a=0, b=0, t="xy"):
```

```
        if t != "xy":
```

```
            t = "polar"
```

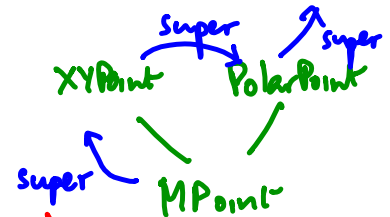
```
        self.ptype = t
```

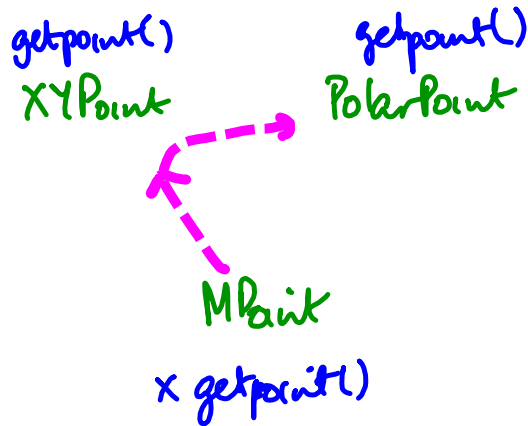
```
        if self.ptype == "xy":
```

```
            super(MPoint, self).__init__(a, b)
```

```
        else
```

```
            super(XYPoint, self).__init__(a, b)
```





Need to check type
& call the
correct getpoint()

Even if an MPoint is
set up using --init--
of PolarPoint,
it remains of type MPoint

"Magic" functions

- `--init--` automatically called when object is created
- `--str--` implicitly called when `str(o)` is used
- `--repr--` also returns a string `"MPoint(3,4)"` ^{`"(3,4)"`}
- `--add--` $p+q \leadsto$ calls `p.--add--(q)`