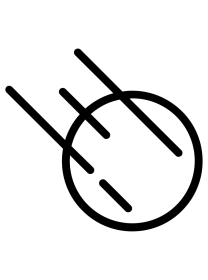
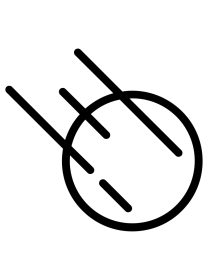


First-Class Patterns as Types

- We already have seen that patterns behave like data types, consider,
  - let x:%integer = v.
- Here the pattern %integer that matches all integer values limits what kind of values can be assigned to the variable x.
- That is precisely what type declarations do!

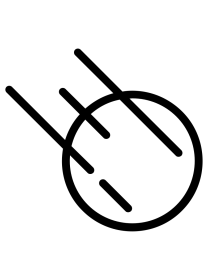


Subtypes

- First-class patterns can be used to define **subtypes of existing types**
- Consider for example,

```
let Pos_Int = pattern %[k if (k is %integer) and (k>0)]%.

let x:*Pos_Int = v.
```

- Here we can treat the pattern Pos_Int as a subtype of the integers, in effect we have
  - Pos_Int < integer

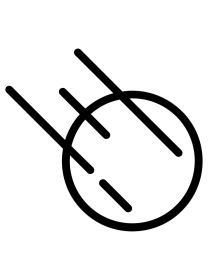


Supertypes

- We can use first class patterns to also define supertypes, consider

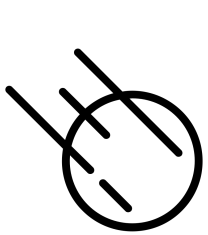
```
let Scalar = pattern %[x if (x is %integer) or (x is %real)]%.

let i:*Scalar = v.
```

- Here the second let statement is only successful if it fulfills the requirements of the pattern Scalar.
- In effect, Scalar acts like a supertype of real and integer
- or more precisely it acts like an **abstract base class** since you since you cannot instantiate a value of type Scalar.



Sub- and Supertypes

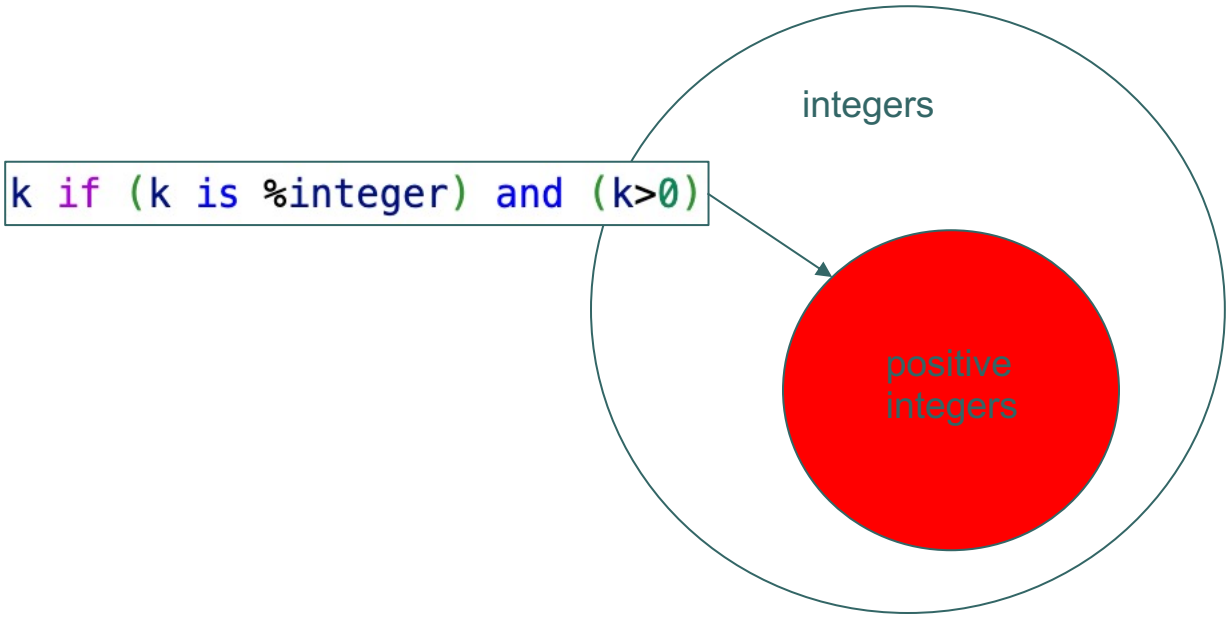
- We use first-class patterns to instantiate both subtypes and supertypes – how do they differ?

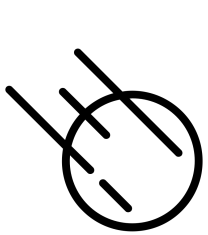


Sub- and Supertypes

- **Subtypes**: the pattern definition adds conditions that **contract** a given data type

```
let Pos_Int = pattern %[k if (k is %integer) and (k>0)]%.
```



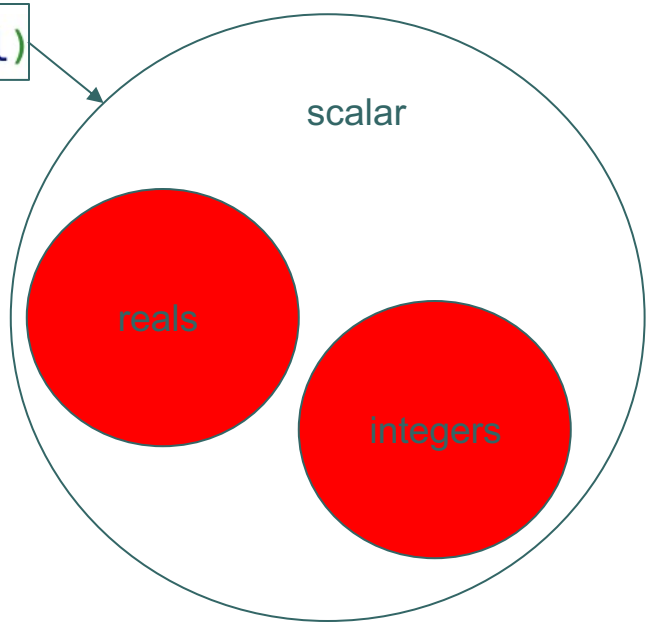


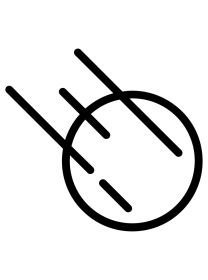
Sub- and Supertypes

- **Supertypes**: the pattern definition **expands** given data types so that the supertype pattern covers more objects than any given data type within the pattern definition.

```
let Scalar = pattern %[x if (x is %integer) or (x is %real)]%.
```

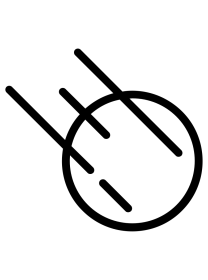
```
x if (x is %integer) or (x is %real)
```





Programming with Patterns as Data Types

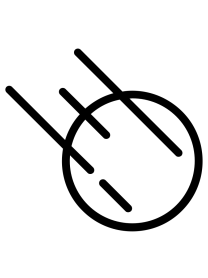
- We an impose a certain amount of type safety with patterns as data types
  - Specification of function domains
  - Type safety for objects using patterns as types in constructors
  - Subtype polymorphism



Function Domains

```
let Pos_Int = pattern %[(x:%integer) if x>0]%.

function fact
   with 0 do
      1
   with n:*Pos_Int do
      n*fact(n-1)
end

assert (fact 3 == 6).
```

ln016/fact.ast



Objects

```
structure Address with
    data street.
    data city.
    data zip.
    function __init__ with (street:%string,city:%string,zip:%string) do
        let this@street = street.
        let this@city = city.
        let this@zip = zip.
    end
end

structure Person with
    data name.
    data profession.
    data address.
    function __init__ with (name:%string,profession:%string,address:%Address) do
        let this@name = name.
        let this@profession = profession.
        let this@address = address.
    end
end

let joe = Person("Joe","Carpenter",Address("532 Main Street","Newport","02840")).
```