Programming I

Lecture 12

Introduction to
Object Oriented Programming

What did we talk about last time?

FILES

BINARY FILES

CSV AND JSON FILES

What will we talk about today?

- Objects
- Classes
- Object construction
- Fields & Methods
- Inheritance
- Polymorphism

What is an object?

a material thing that can be seen and touched

- A **sequence of bytes** stored in memory
  - **data**
  - **code**

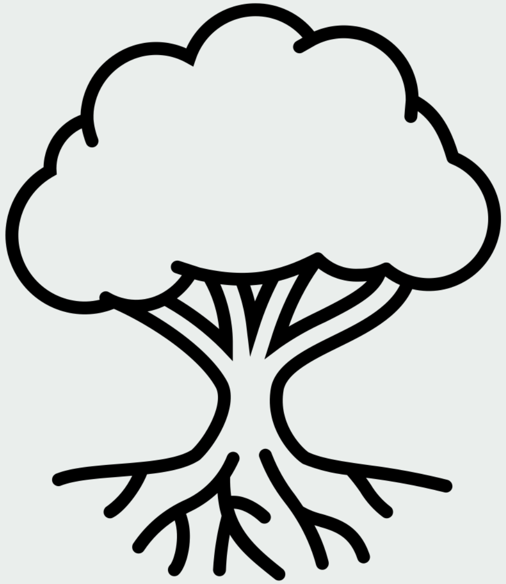
Example objects in the world

Example objects in Python

- Strings, Lists, Tuples, Sets, Dictionaries
- File Handler
- CSV Reader
- Most things in Python are objects

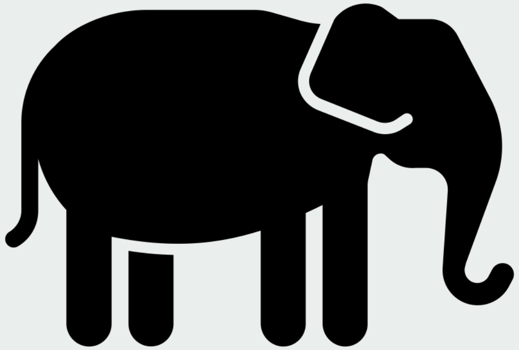
What makes a tree an object?

- A tree occupies some space
- A tree has a behaviour:
  - “eats” carbon dioxide, sunlight, water, nutrients
  - excretes oxygen
  - Grows
- There is a recipe for creating a new tree



What makes an elephant an object?

- An elephant occupies some space
- An elephant has a behaviour:
  - “eats” **tree** leaves, water, oxygen
  - excretes nutrients
  - Grows
- There is a recipe for creating a new elephant



What makes a list an object?

- You have a recipe for constructing new lists
- You can call functions that:
  - **use** the **state** (data) of a list (count, index)
  - **alter** the **state** of a list (append, extend, sort, reverse)

How can we define this recipe?

Classes

- define how objects are created
  - object state
  - class state
- define functions:
  - object methods
  - class methods

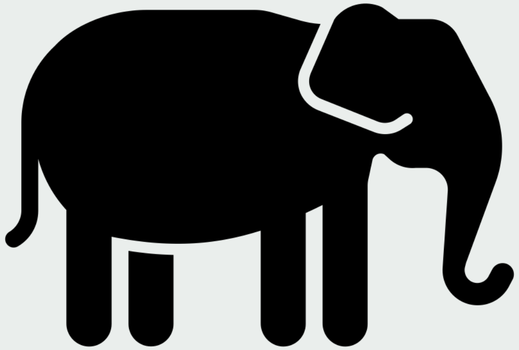
Modelling an elephant (construction)

```
class Elephant:
    def __init__(self, name):
        self.name = name
```

Constructor definition

```
e = Elephant("Dumbo")
print(e.name)
```

Constructor call



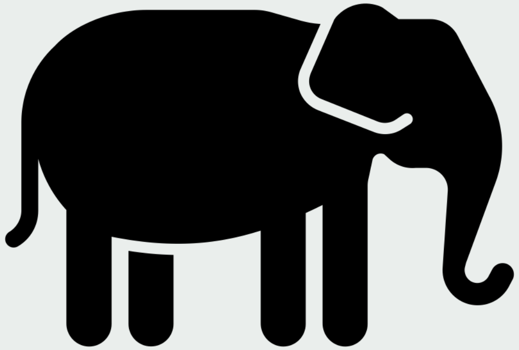
Modelling an elephant (construction)

```
class Elephant:
    def __init__(self, name):
        self.name = name
```

self = current object

```
e = Elephant("Dumbo")
print(e.name)
```

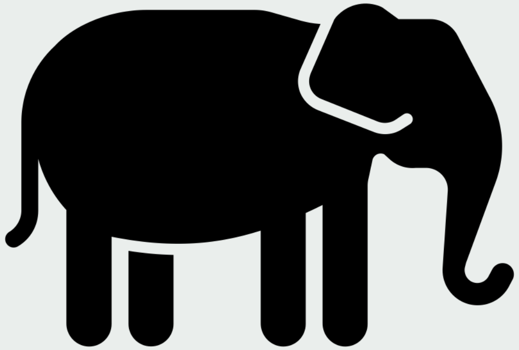
No **self** reference here



Modelling an elephant (object fields)

```
class Elephant:
    def __init__(self, name):
        self.name = name

e = Elephant("Dumbo")
print(e.name)
```

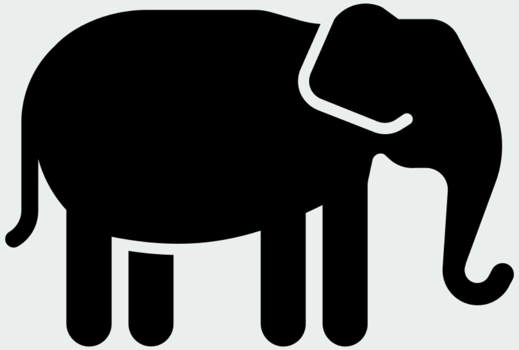
Accessing the **name** field of object **e**



Modelling an elephant (object fields)

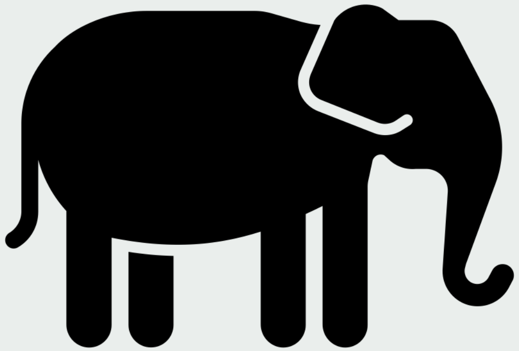
```
class Elephant:
    def __init__(self, name):
        self.name = name

e = Elephant("Dumbo")
e.name = "Jumbo"
print(e.name)
```

Setting the **name** field of object **e**



Modelling an elephant (more fields)

```
class Elephant:
    def __init__(self, name, age, weight):
        self.name = name
        self.age = age
        self.weight = weight

e = Elephant("Dumbo", 0, 100)
print(e.name, e.age, e.weight)
```



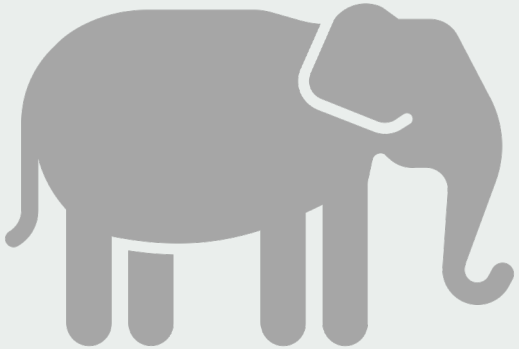
Modelling an elephant (class fields)

```
class Elephant:
    color = "gray"
    daily_food_intake = 100 # kg of leaves per ton of weight

    def __init__(self, name, age, weight):
        self.name = name
        self.age = age
        self.weight = weight

e = Elephant("Dumbo", 0, 100)
print(Elephant.color)
print(Elephant.daily_food_intake)
print(e.color)
```

Class fields

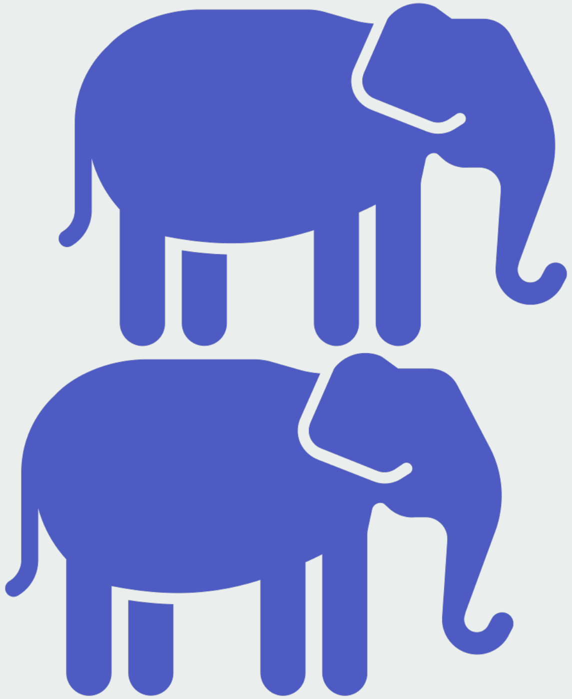
Access **Class fields** through class name

Access **Class fields** through **object**



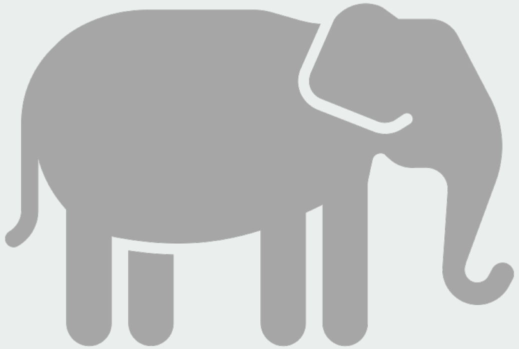
Modelling an elephant (class fields)

```
e = Elephant("Dumbo", 0, 100)
e2 = Elephant("Jumbo", 2, 1200)

Elephant.color = "blue"

print(e.color, e2.color)
```



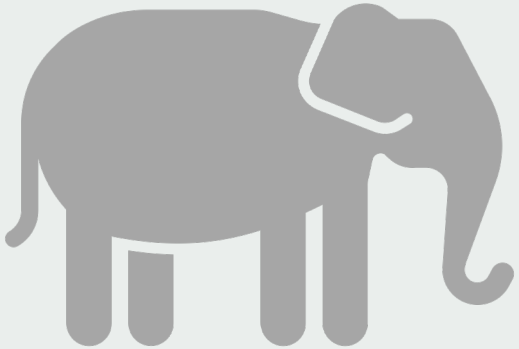
Modelling an elephant (methods)

```
class Elephant:
    def __init__(self, name, age, weight):
        ...
        self.food_today = 0

    def eat(self, kg):
        self.food_today += kg

e = Elephant("Dumbo", 0, 100)

e.eat(10)
e.eat(20)
assert e.food_today == 30
```



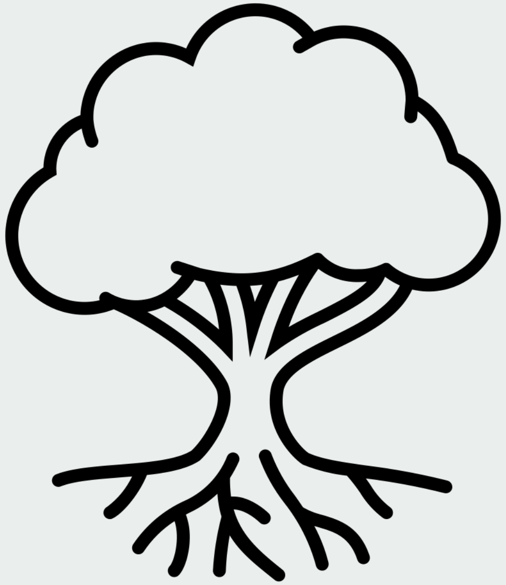
Modelling an elephant (methods)

```
class Elephant:
    ...
    def grow(self):
        daily = (Elephant.daily_food_intake / 1000) * self.weight
        if self.food_today >= daily:
            self.weight += 0.2 * self.weight
            self.food_today -= daily

e = Elephant("Dumbo", 0, 100)

e.eat(10)
e.grow()
print(e.weight)
```



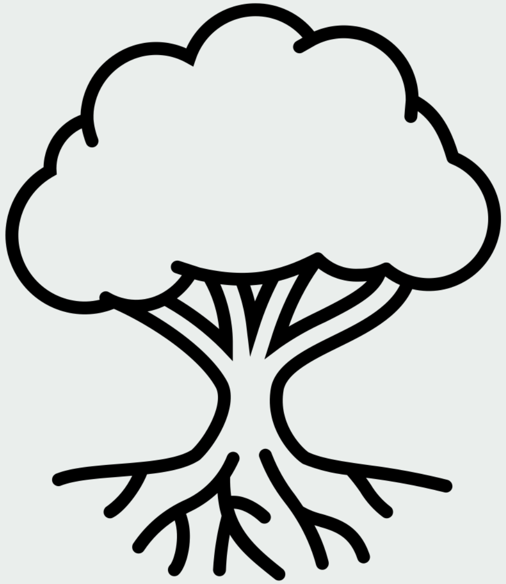
Modelling a tree

```
class Tree:
    MIN_LEAF = 50
    LEAF_WEIGHT = 0.01 # kg (10g per leaf)
    def __init__(self, species, age):
        self.species = species
        self.age = age
        self.branches = 1 + 2 ** age
        self.leaves = self.branches * Tree.MIN_LEAF

t = Tree("Acacia", 10)
print(t.branches)
```



Making the tree grow

```
class Tree:
    ...

    def grow(self):
        self.leaves += self.branches  # one leaf per branch
```



Displaying user-defined objects

```
t = Tree("Acacia", 2)
e = Elephant("Dumbo", 1, 500)
print(t)
print(e)
```

```
<__main__.Tree object at 0x10c0c8f28>
<__main__.Elephant object at 0x10c0c8eb8>
```

The __str__ magic method

```
class Tree:
    …

    def __str__(self):
        return "Tree{{spec:{}, age:{}, branches:{}, leaves:{}}}”
                     .format(self.species, self.age, self.branches, self.leaves)

class Elephant:
    …

    def __str__(self):
        return "Elephant{{name:{}, age:{}, weight:{}, food:{}}}”
                     .format(self.name, self.age, self.weight, self.food_today)
```

```
t = Tree("Acacia", 2)
e = Elephant("Dumbo", 1, 500)
print(t)
print(e)
```

```
Tree{spec:Acacia, age:2, branches:5, leaves:250}
Elephant{name:Dumbo, age:1, weight:500, food:0}
```

Displaying user-defined objects

```python
t = Tree("Acacia", 2)
e = Elephant("Dumbo", 1, 500)
print([t]) # list of trees
print([e]) # list of elephants
```

```
[<__main__.Tree object at 0x1021a4eb8>]
[<__main__.Elephant object at 0x1021a4be0>]
```

The __repr__ magic method

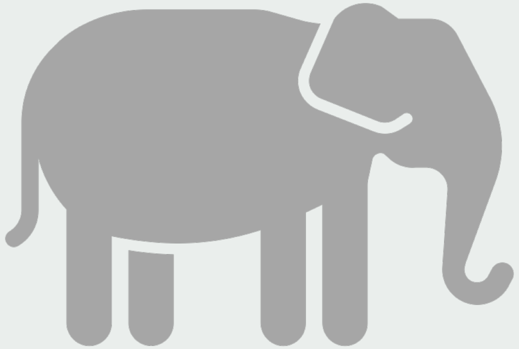
```
def __repr__(self):
    return self.__str__()
```

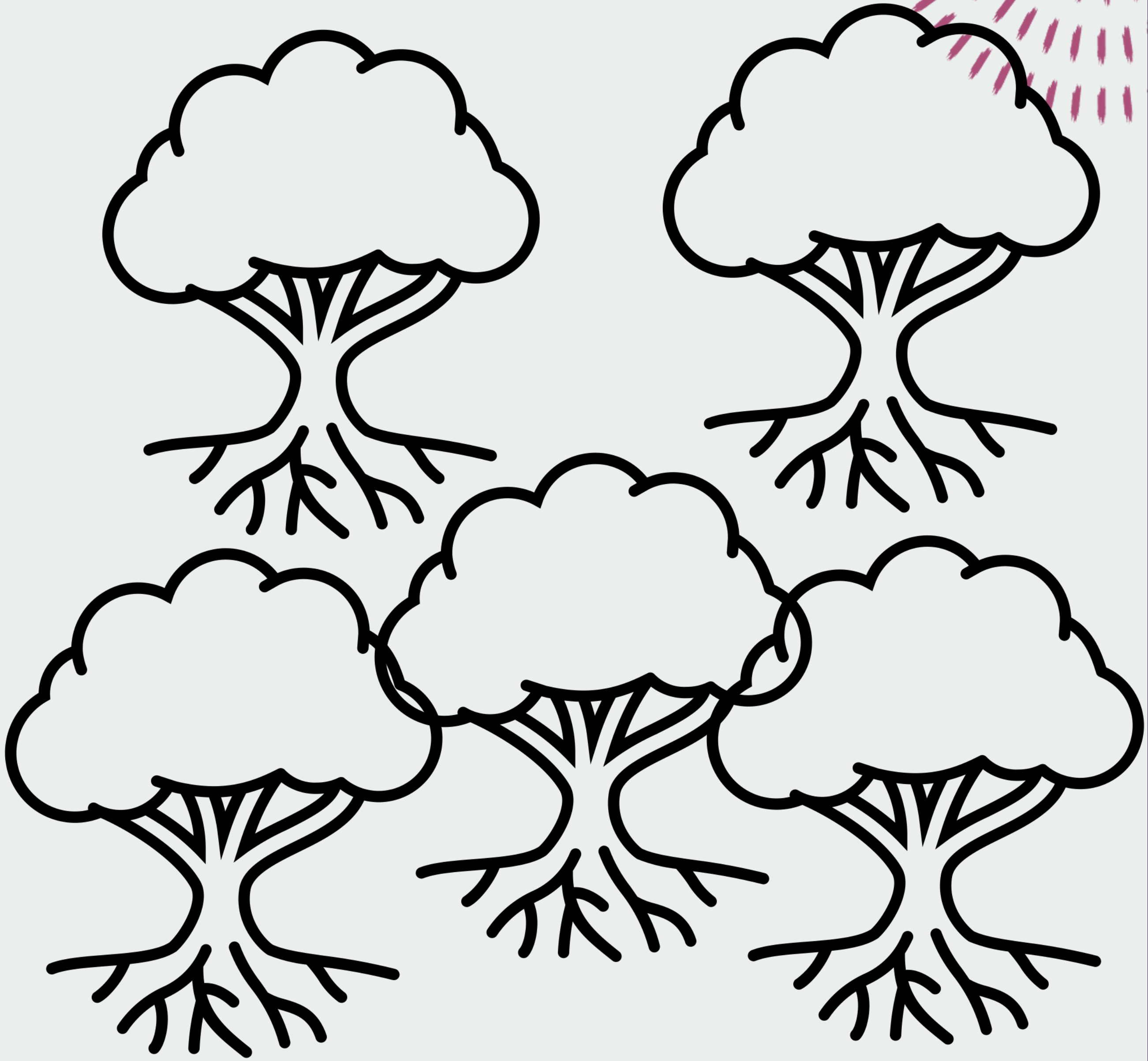
All magic methods:
https://docs.python.org/3/reference/datamodel.html

```
t = Tree("Acacia", 2)
e = Elephant("Dumbo", 1, 500)
print([t]) # list of Trees
print([e]) # list of Elephants
```

```
[Tree{spec:Acacia, age:2, branches:5, leaves:250}]
[Elephant{name:Dumbo, age:1, weight:500, food:0}]
```

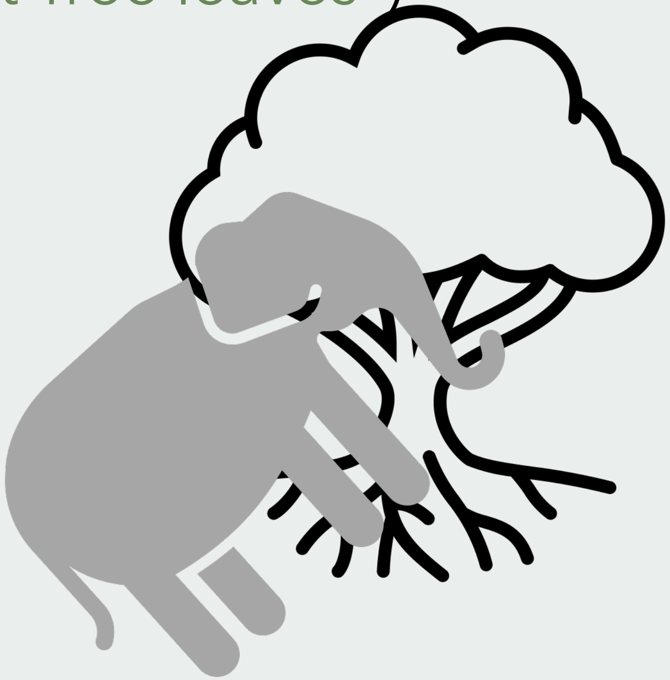
The elephant is hungry…





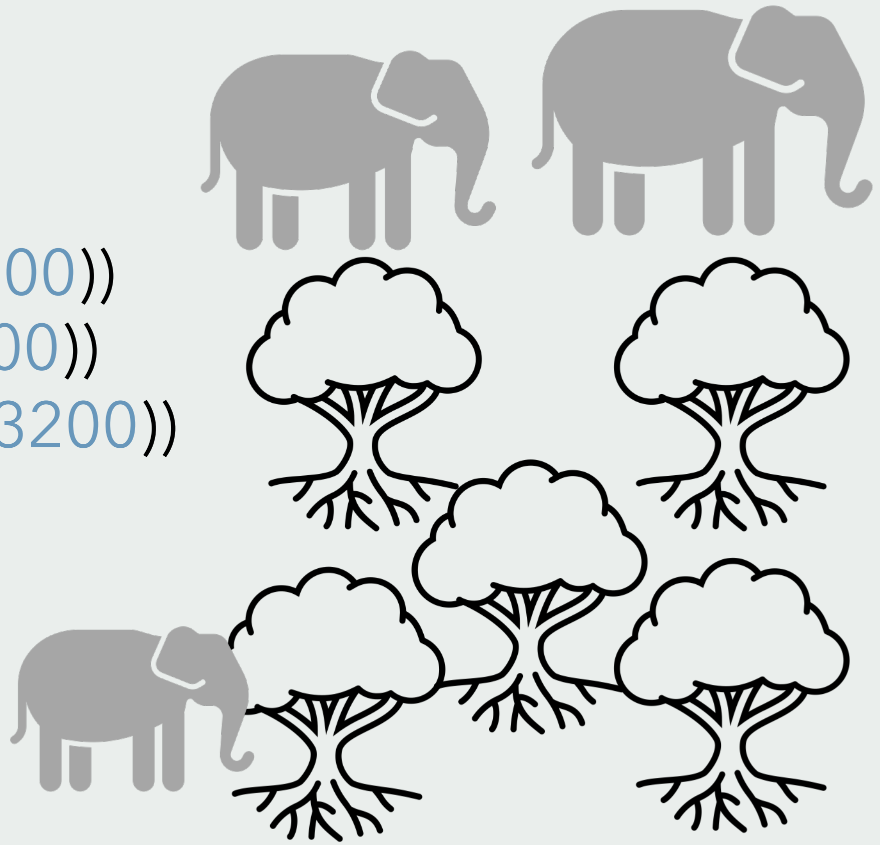
Eating some leaves

```
class Elephant:
    ...
    def eat(self, tree):

        if not isinstance(tree, Tree):
            raise ValueError("An elephant can only eat Tree leaves")
        # eat 10% of leaves
        x = tree.leaves / 10
        tree.remove_leaves(x)
        self.food_today += x * Tree.LEAF_WEIGHT
```



Creating some objects

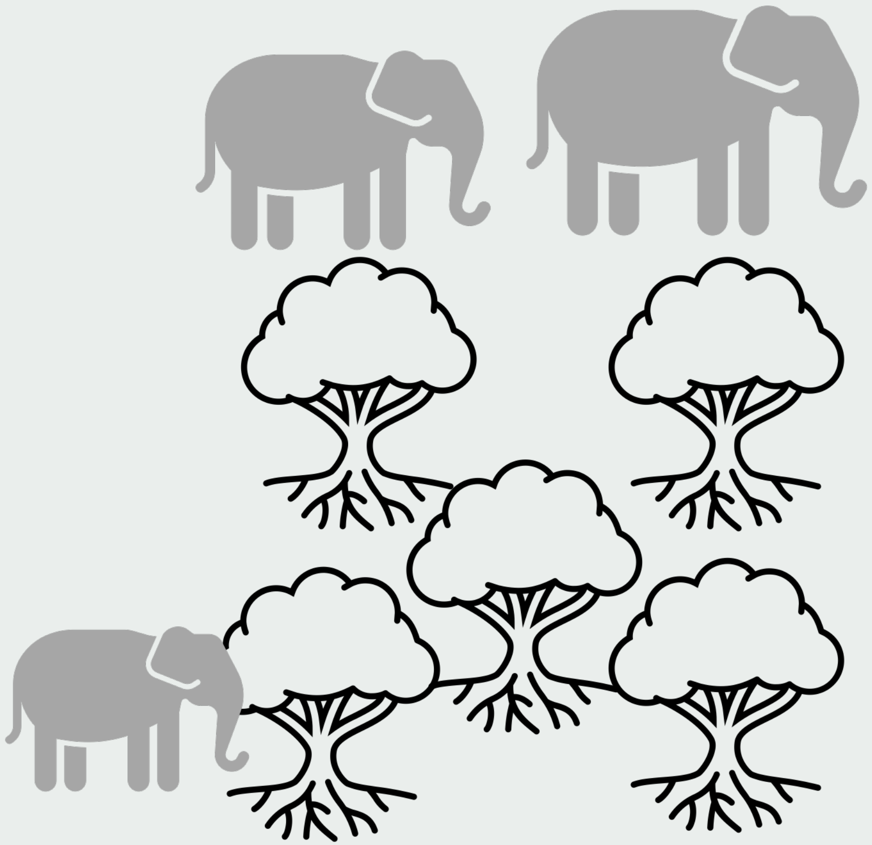
```
trees = []
elephants = []

for i in range(10):
    trees.append(Tree("Acacia", i))

elephants.append(Elephant("Dumbo", 1, 600))
elephants.append(Elephant("Jumbo", 2, 900))
elephants.append(Elephant("Zumbo", 10, 3200))
```



Letting them rumble

```
DAYS = 100

for i in range(DAYS):
    for t in trees:
        t.grow()
    for e in elephants:
        x = random.randint(0, len(trees)-1)
        e.eat(trees[x])
        e.grow()

print(elephants)
```

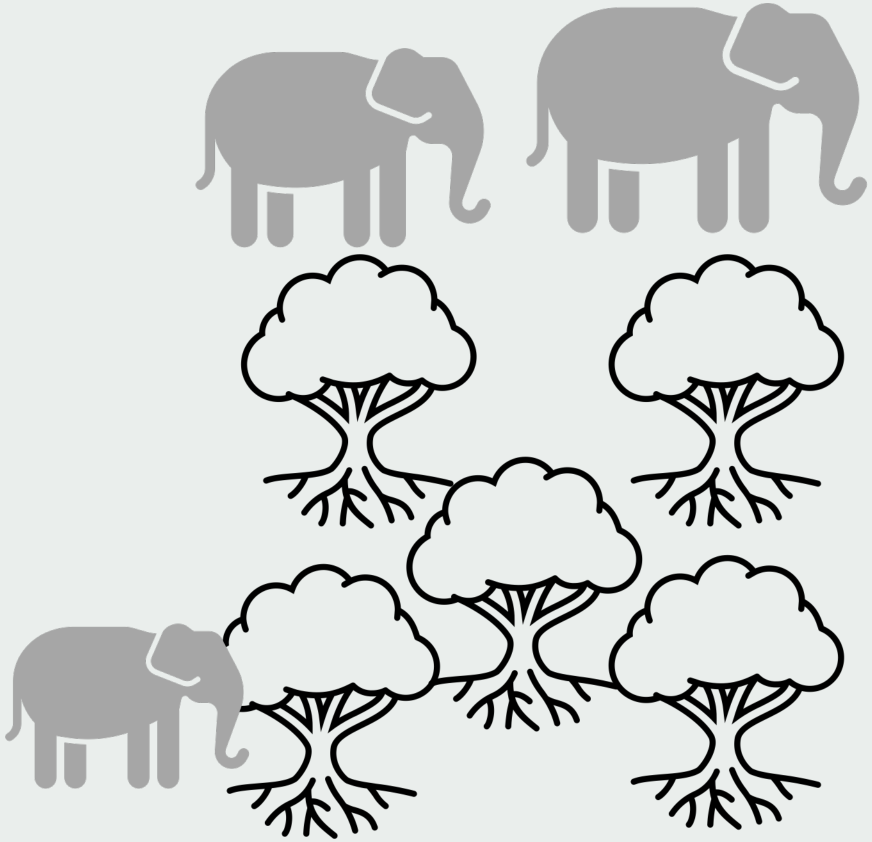
```
Elephant{name:Dumbo, age:1, weight:1036.8, food:66.079}
Elephant{name:Jumbo, age:2, weight:1866.24, food:144.635}
Elephant{name:Zumbo, age:10, weight:3200, food:271.946}
```



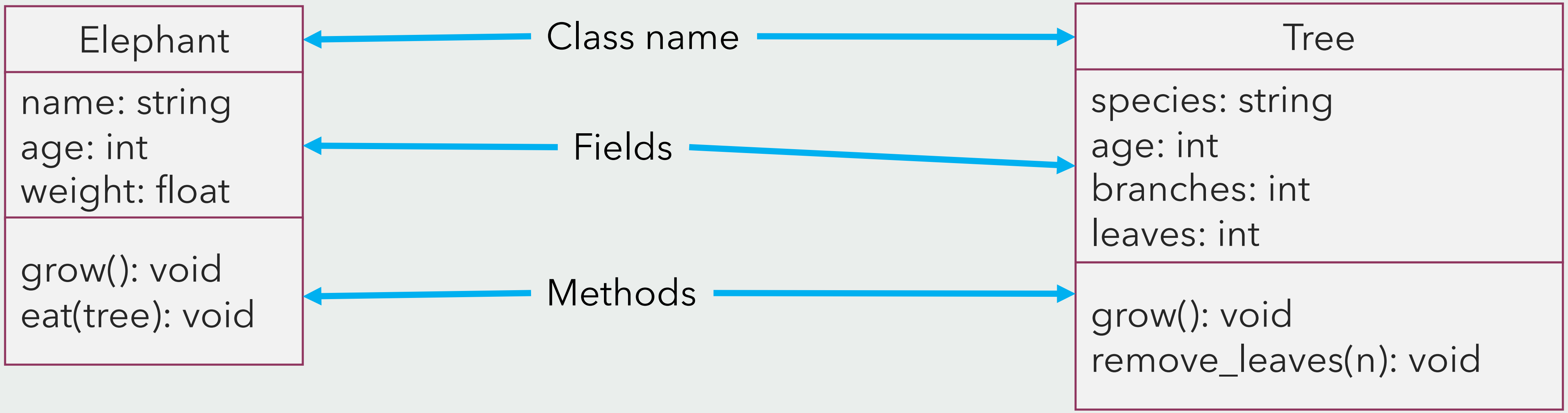
Can we make an abstraction?

```
DAYS = 100

for i in range(DAYS):
    for t in trees:
        t.grow()
    for e in elephants:
        x = random.randint(0, len(trees)-1)
        e.eat(trees[x])
        e.grow()

print(elephants)
```

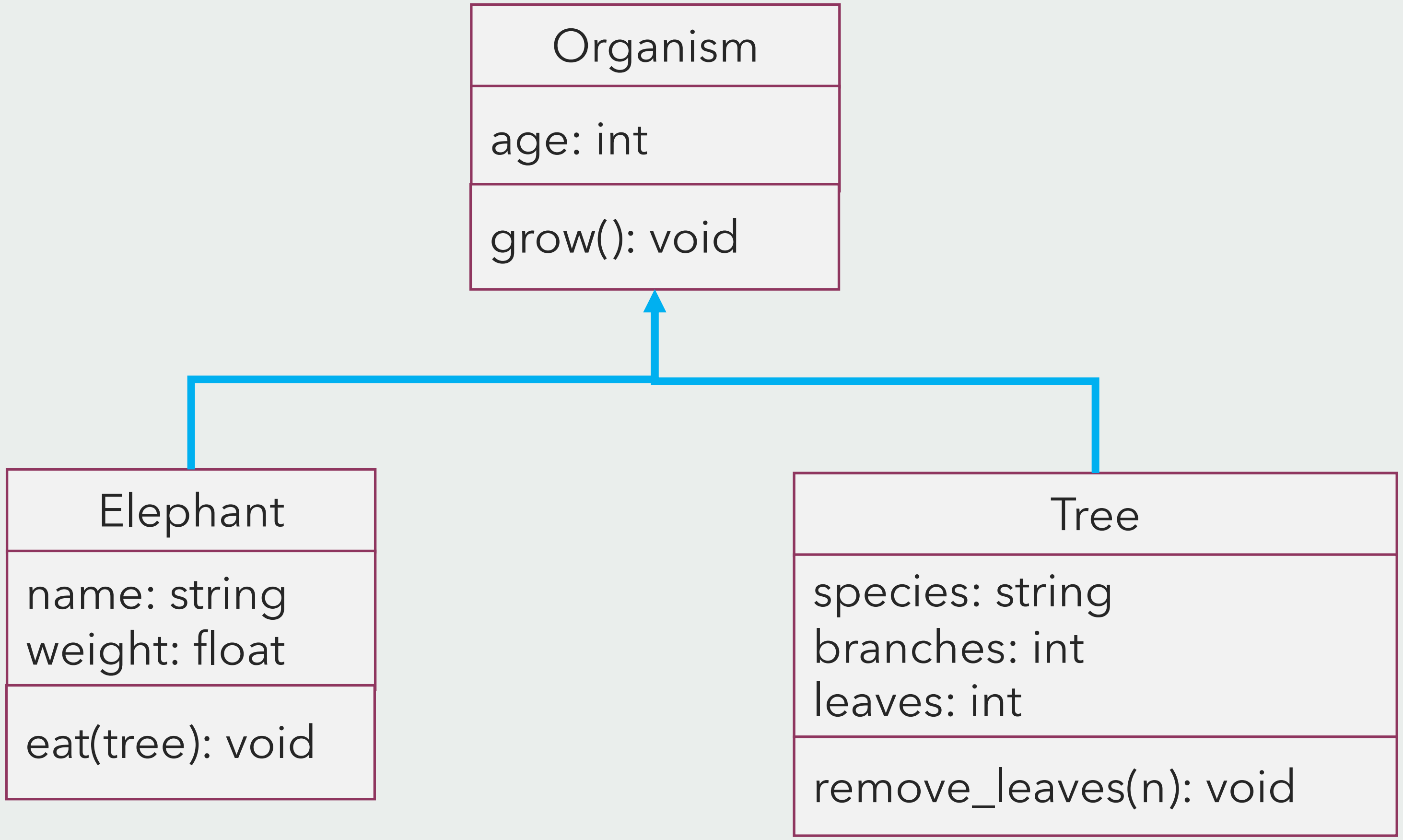
Function called for both types of objects



Class diagrams



Inheritance



Organism

age: int

grow(): void

Elephant

name: string
weight: float

eat(tree): void

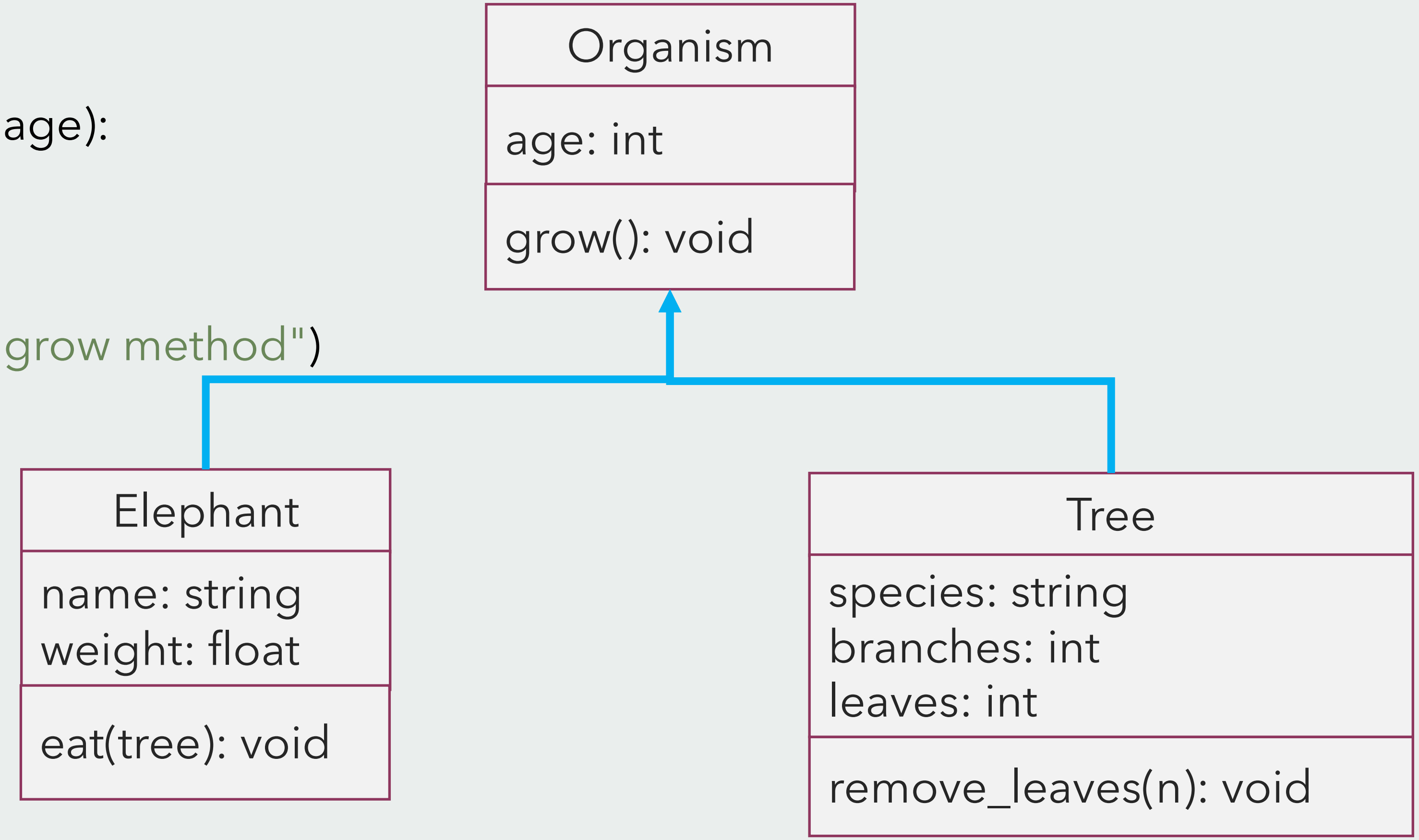
Tree

species: string
branches: int
leaves: int

remove_leaves(n): void

Inheritance

```
class Organism:
    def __init__(self, age):
        self.age = age

    def grow(self):
        print("Sample grow method")
```



Organism

age: int

grow(): void

Elephant

name: string
weight: float

eat(tree): void

Tree

species: string
branches: int
leaves: int

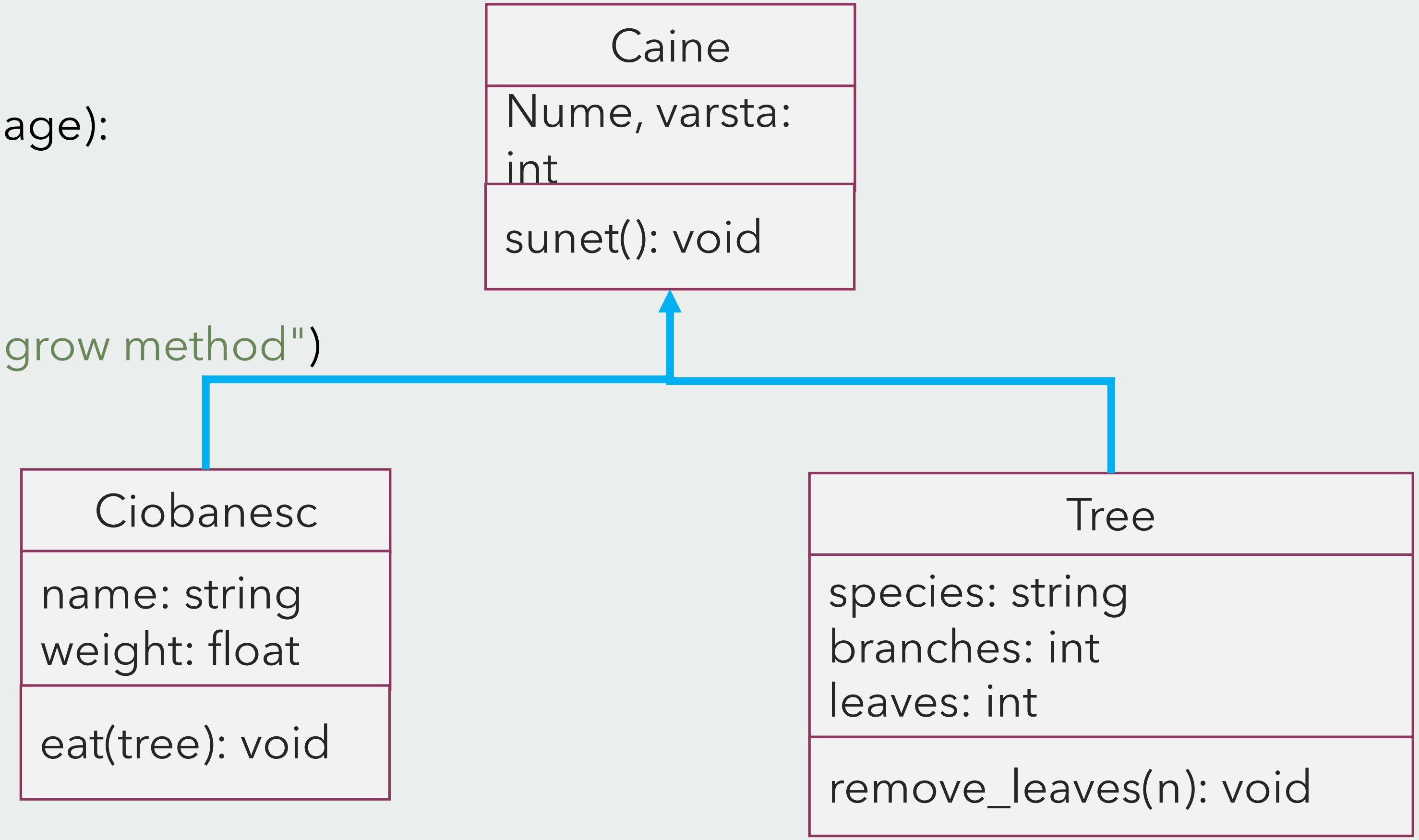
remove_leaves(n): void

```
class Elephant(Organism):

    def __init__(self, name, age, weight):
        super().__init__(age)
        …
```

```
class Tree(Organism):

def __init__(self, species, age):
        super().__init__(age)
        …
```

Inheritance

```
class Organism:
    def __init__(self, age):
        self.age = age

    def grow(self):
        print("Sample grow method")
```



Caine

Nume, varsta: int

sunet(): void

Ciobanesc

name: string
weight: float

eat(tree): void

Tree

species: string
branches: int
leaves: int

remove_leaves(n): void

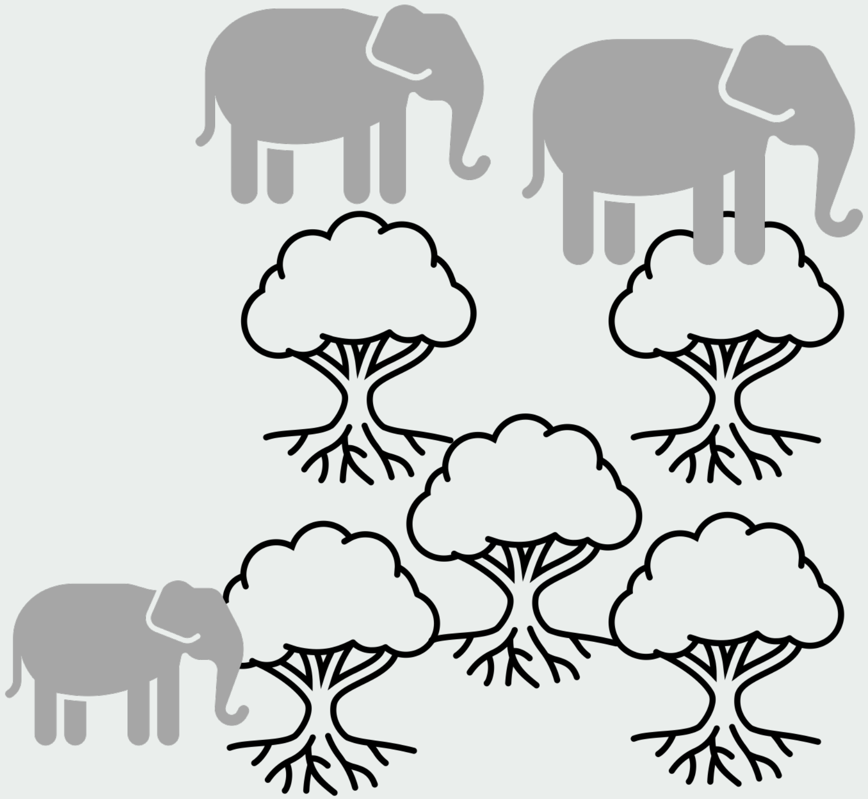
```
class Elephant(Organism):

    def __init__(self, name, age, weight):
        super().__init__(age)
        …
```

```
class Tree(Organism):

def __init__(self, species, age):
      super().__init__(age)
      …
```

Attribute inheritance

```
t = Tree("Acacia", 2)
e = Elephant("Dumbo", 1, 500)
o = Organism(2)

print(t.age, e.age, o.age)
```

Sub-classes have inherited this field



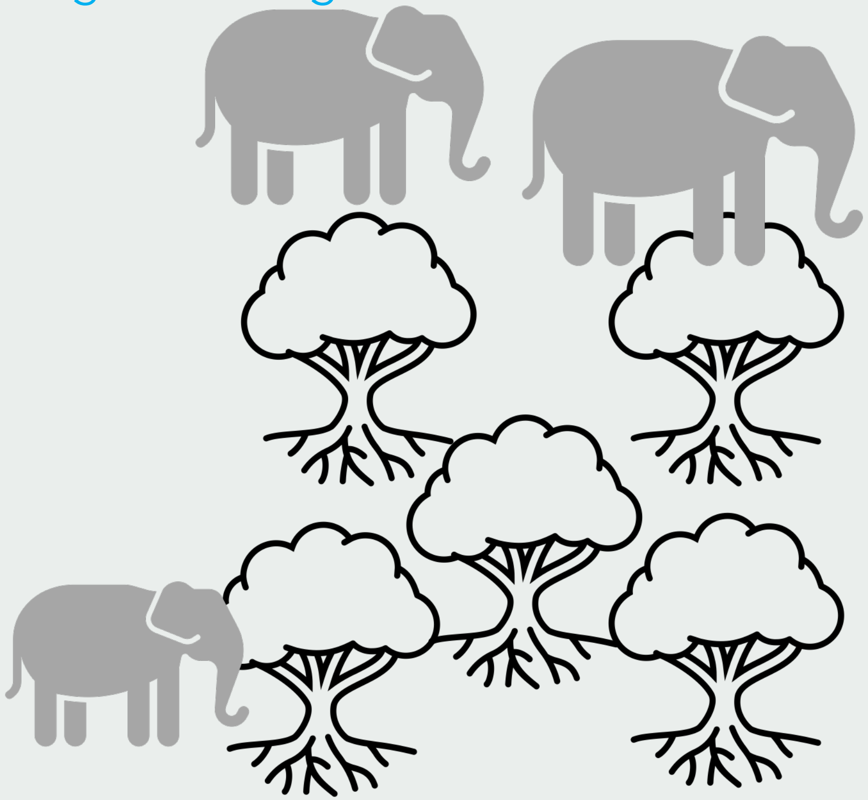
Method override

```
t = Tree("Acacia", 2)
e = Elephant("Dumbo", 1, 500)
o = Organism(2)


t.grow()
e.grow()
o.grow()
```

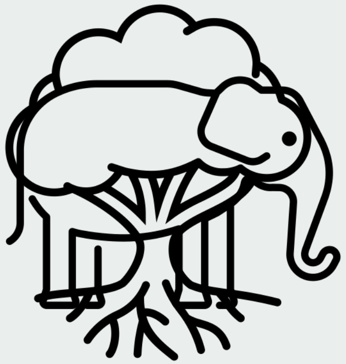
Increases number of leaves

Increases weight if enough food eaten

Prints 'Sample grow method'



Polymorphism (gr, many shapes)



```
jungle = []
for i in range(10):
    jungle.append(Tree("Acacia", i+2))

jungle.append(Elephant("Dumbo", 1, 600))
jungle.append(Elephant("Jumbo", 2, 900))
jungle.append(Elephant("Zumbo", 10, 3200))


for i in range(DAYS):
    for o in jungle:
        o.grow()
        if isinstance(o, Elephant):
            x = random.randint(0, len(jungle)-1)
            if isinstance(jungle[x], Tree):
                o.eat(jungle[x])
```

Polymorphism: the grow() function is different depending on the actual object

Check if object is an Elephant

Check if the other object is a Tree

</Programming I>

That’s all folks!

Feedback

https://forms.gle/EcpQ75n4Qff7VV3E8