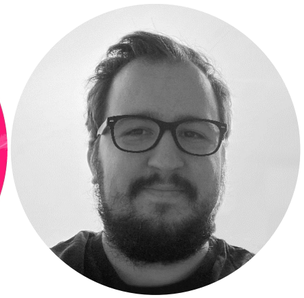
Classes and Object-Oriented Programming in Python 3

Everything Is an Object



Mateo Prigl
Software Developer

Classes and Object-Oriented Programming in Python 3

Version Check

Version Check

This course was created by using:

- Python 3.11

Version Check

This course is 100% applicable to:

- Python 3.10 through Python 3.11

Prerequisites

What is Python?

How to run Python scripts?

Install and configure Python for development

Fundamentals of the Python language

How to import code from other modules?

Overview

Intended audience

- Learners familiar with the basics of the Python language

You can code along with me

- Code from lessons can be found inside of "Exercise Files"

Outcome

- Familiarity with object-oriented programming in Python
- Classes, objects, encapsulation, inheritance, data classes and so on

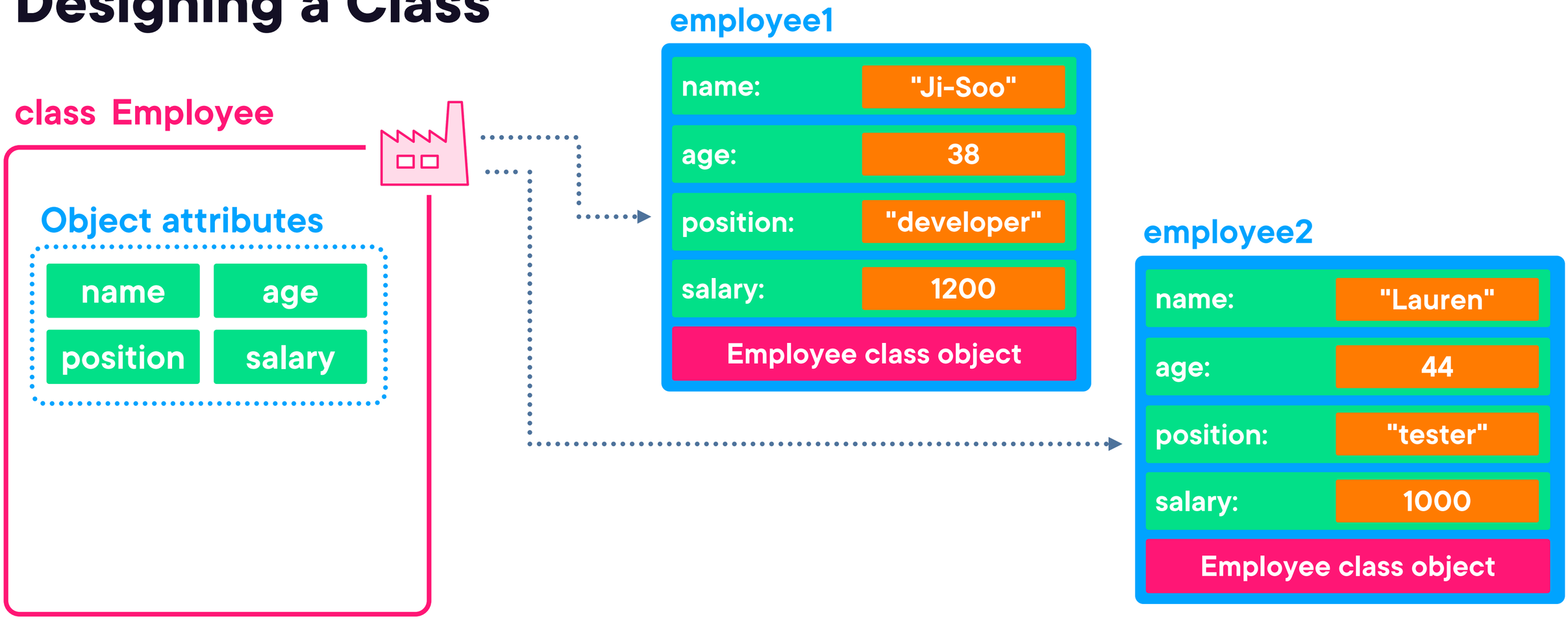
Why do we need object-oriented programming?

Demo

Example:

- Plan out how to structure data for a new application
- Company needs to manage its employees and projects

Designing a Class




```
employee1 = Employee("Ji-Soo", 38, "developer", 1200)
employee2 = Employee("Lauren", 44, "tester", 1000)
```

Designing a Class

class Employee

Object attributes

name	age
position	salary

employee1

name:	"Cynthia"
age:	38
position:	"developer"
salary:	1200

Employee class object

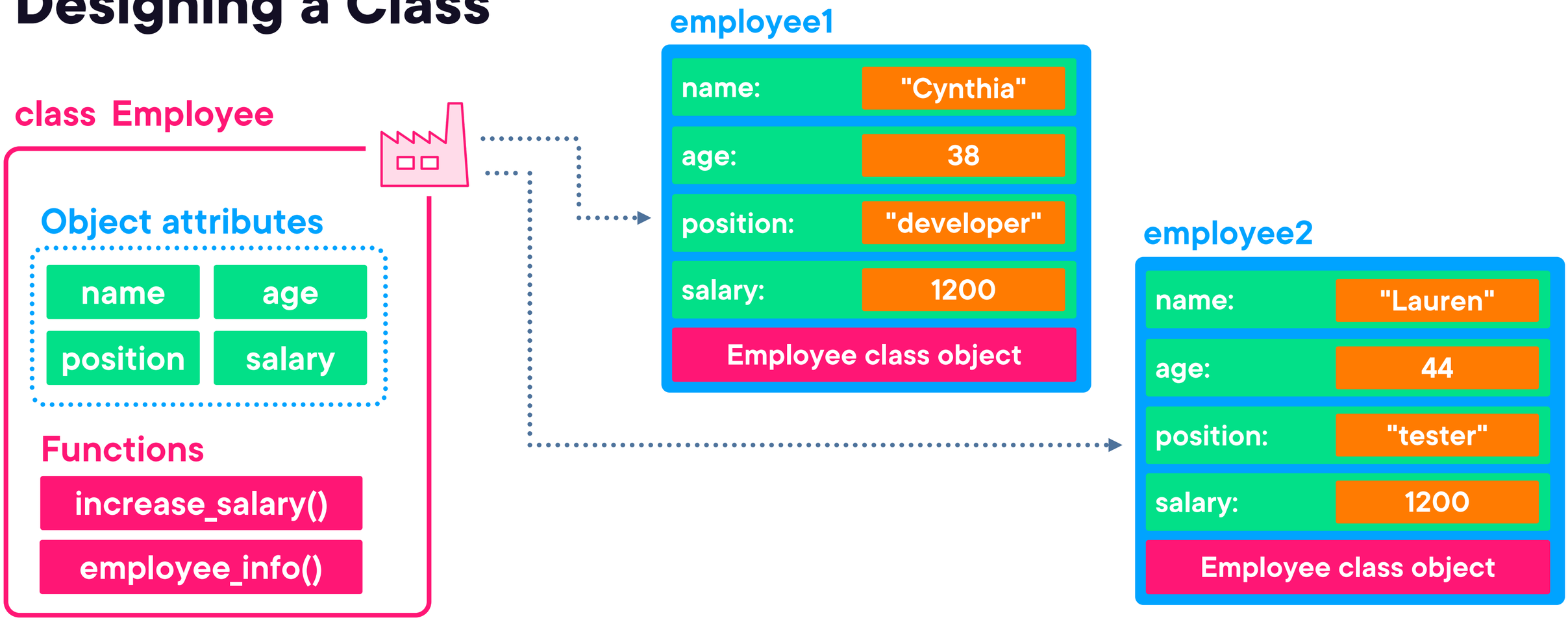
employee2

name:	"Lauren"
age:	44
position:	"tester"
salary:	1000

Employee class object

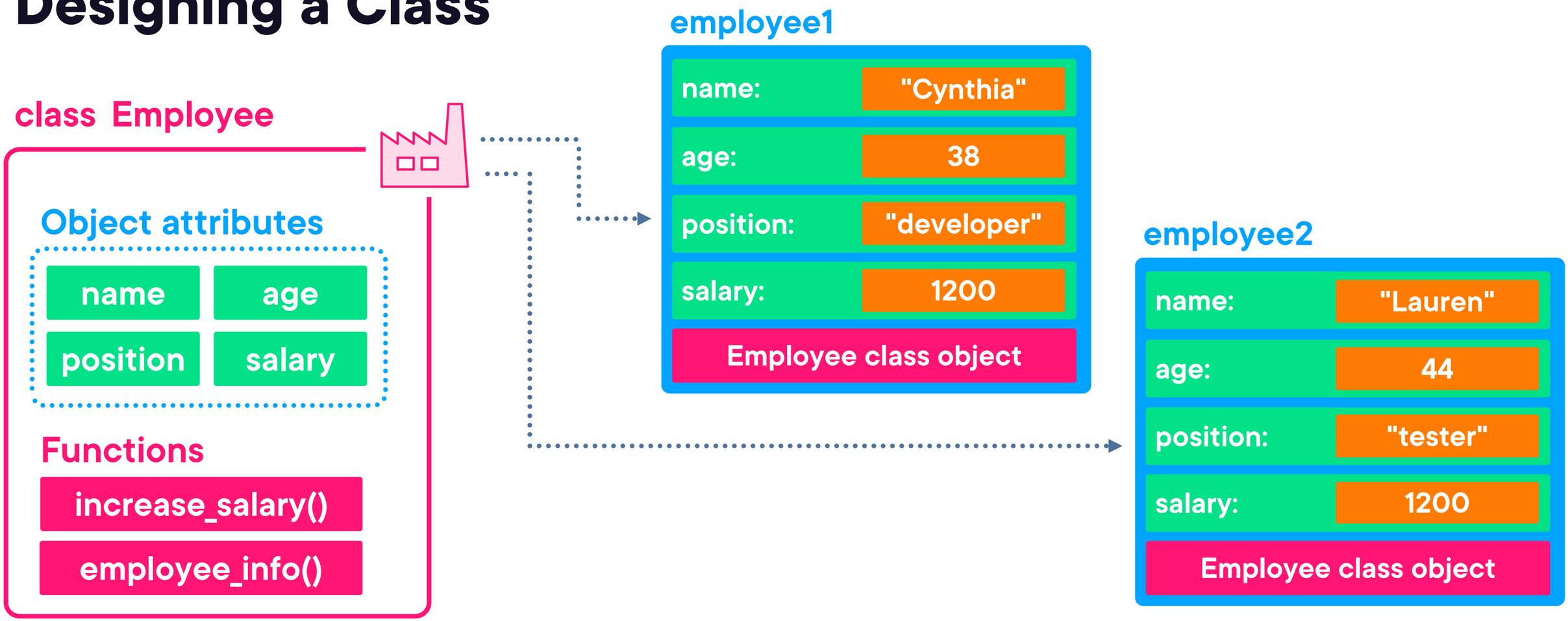
```
print(employee1.name)          # Output: "Ji-Soo"
employee1.name = "Cynthia"
print(employee1.name)          # Output: "Cynthia"
```

Designing a Class



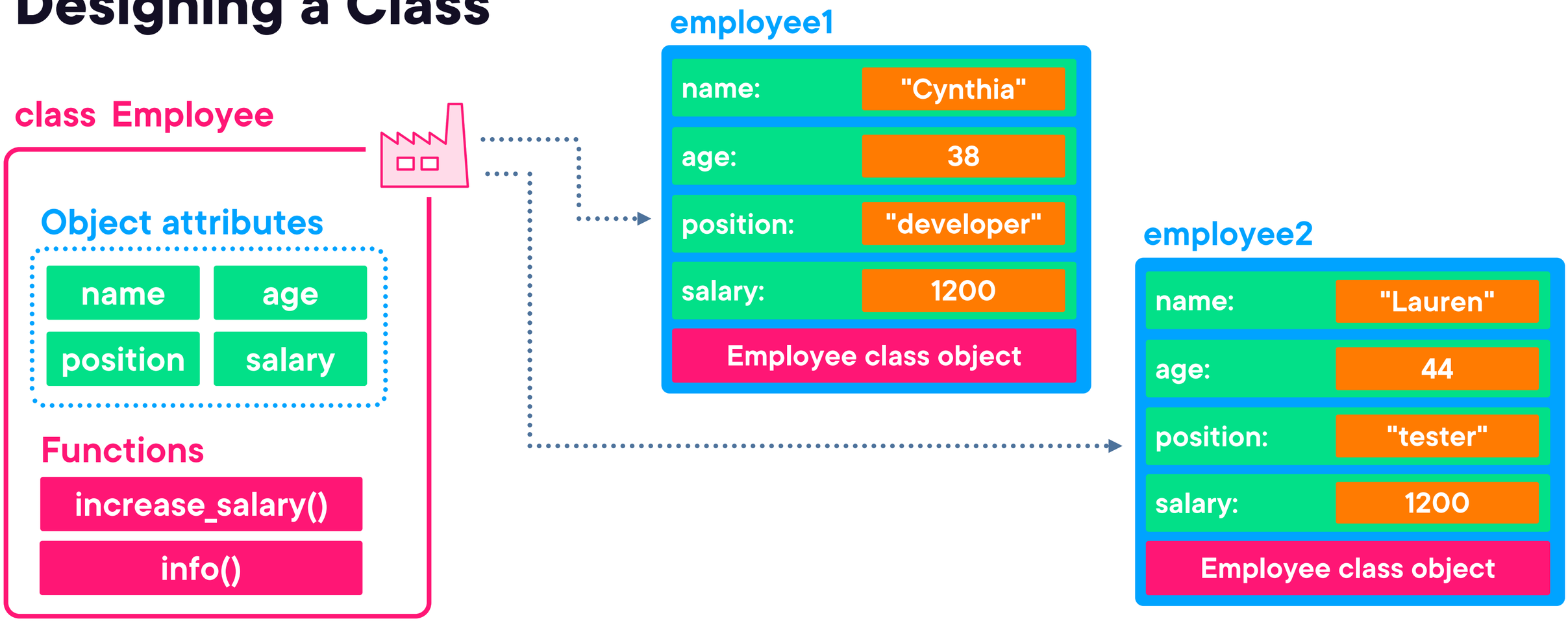

```
employee2.increase_salary(20)
employee2.employee_info()   # Output: Lauren is 44 years old. Employee \
is a tester with the salary of $1200
```

Designing a Class



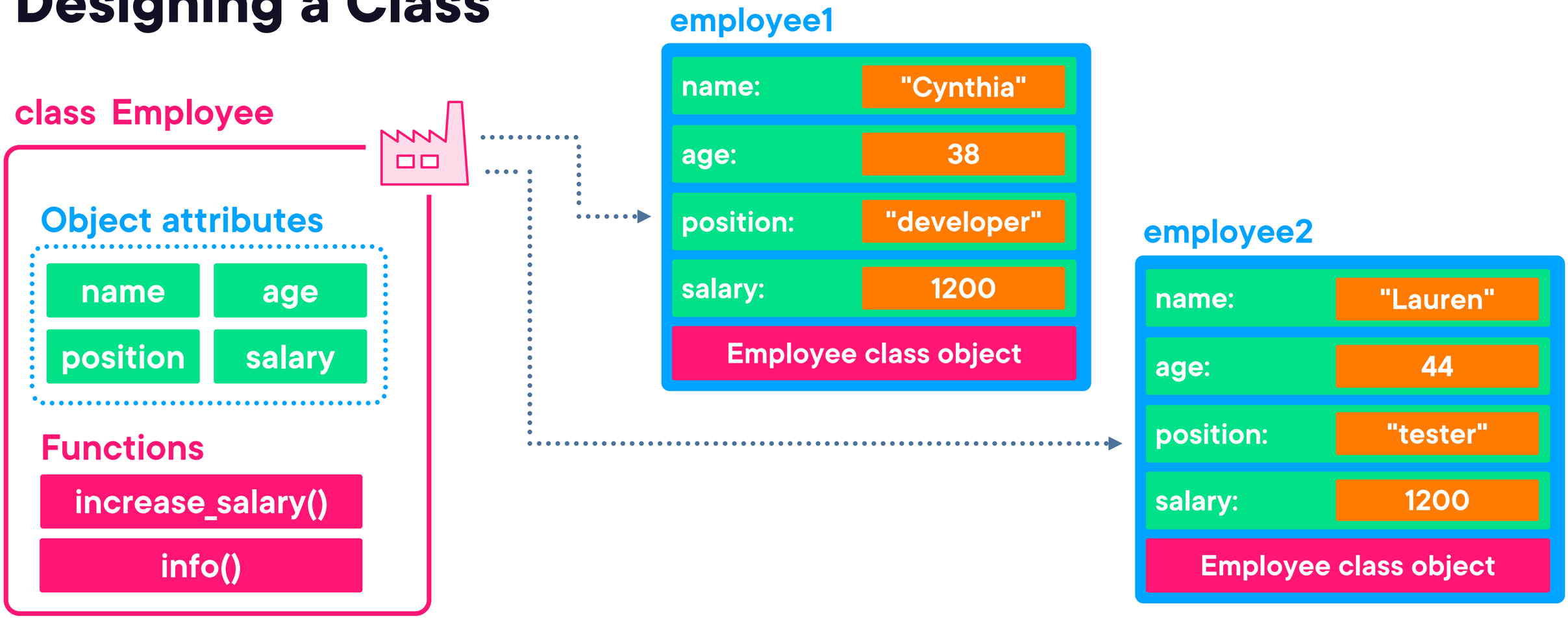

```
employee2.increase_salary(20)
employee2.employee_info()   # Output: Lauren is 44 years old. Employee \
is a tester with the salary of $1200
```

Designing a Class




```
employee2.increase_salary(20)
employee2.info()   # Output: Lauren is 44 years old. Employee \
is a tester with the salary of $1200
```

Designing a Class




```
print(employee2.__class__)    # Output: <class '__main__.Employee'>
```

Should you utilize object-oriented programming?

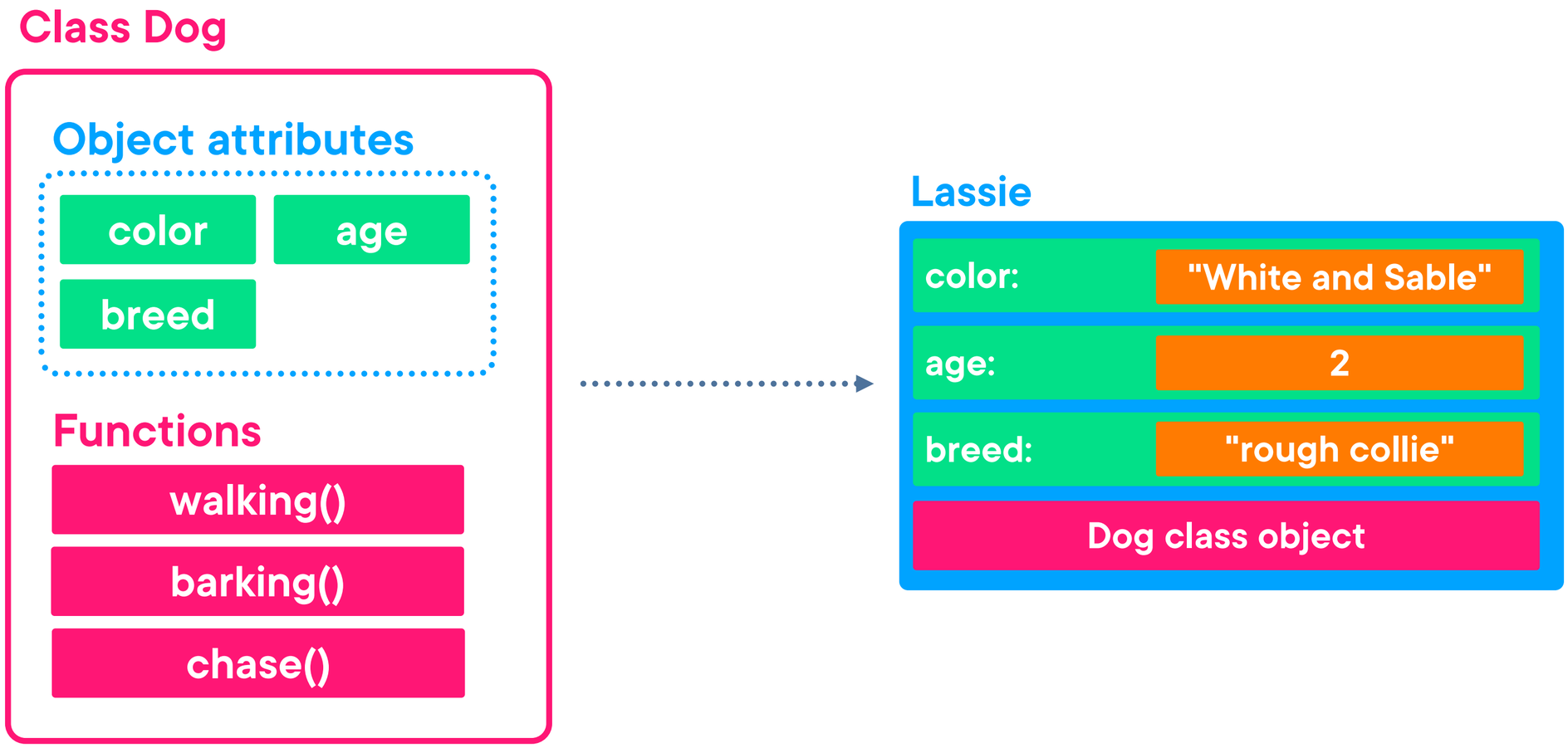
Not the best choice for all types of projects

Great way to split your code in smaller logical units

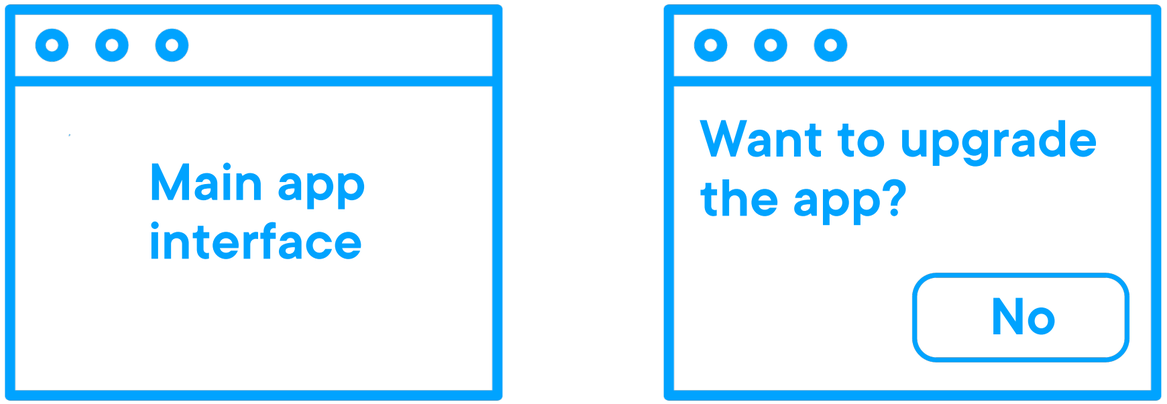
OOP paradigm is implemented in a lot of other programming languages

Model your code around real-world entities

Modeling Classes After Real World Abstractions



Using Packages That Rely on OOP




```
from some_gui_package import Window, Button

win1 = Window("Main app interface")
win2 = Window("Want to update the app?")
btn = Button("No")

if btn.clicked:
    win2.close_window()
```

Summary

Up Next:

Instantiating Custom Classes