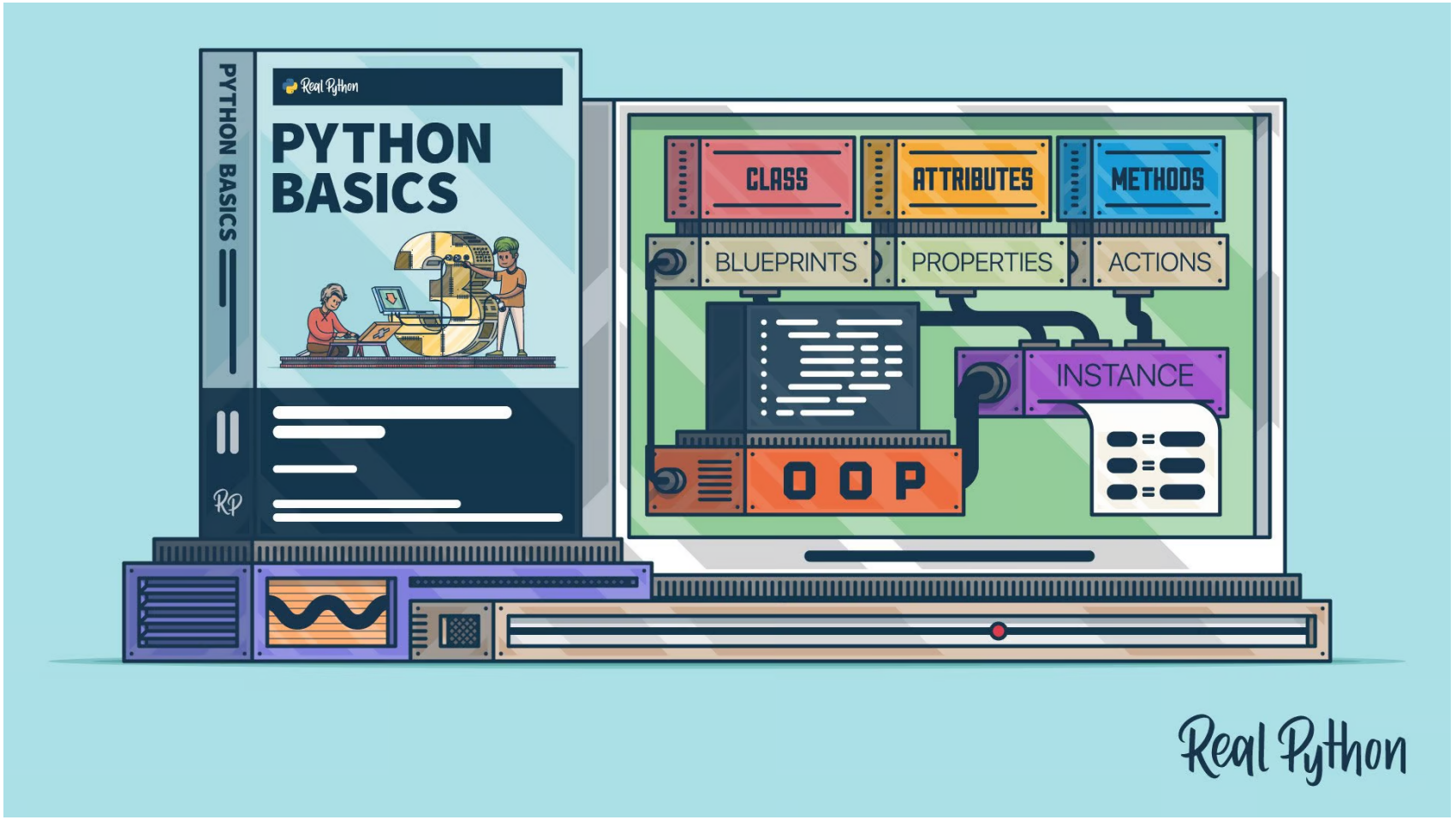
Python Basics Exercises: Object-Oriented Programming

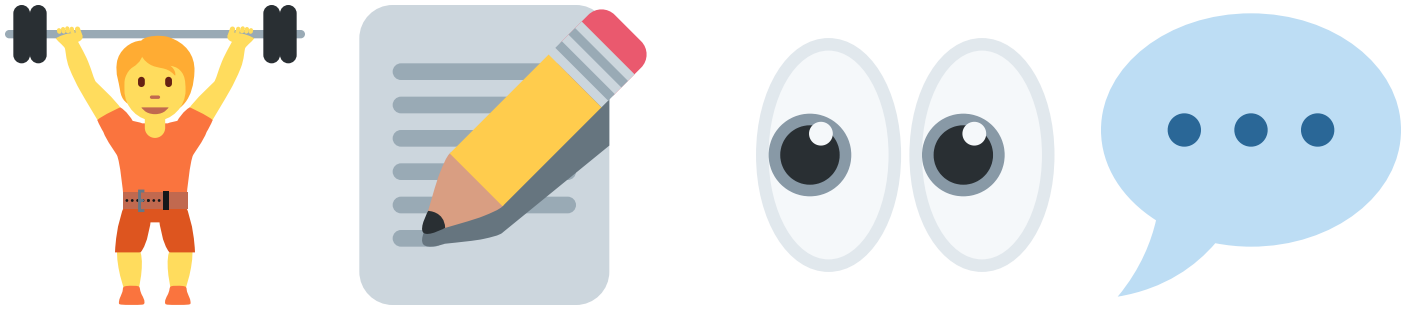


Real Python Exercises Course

Real Python Exercises Course

Real Python Exercises Course

Real Python Exercises Course



Real Python Exercises Course

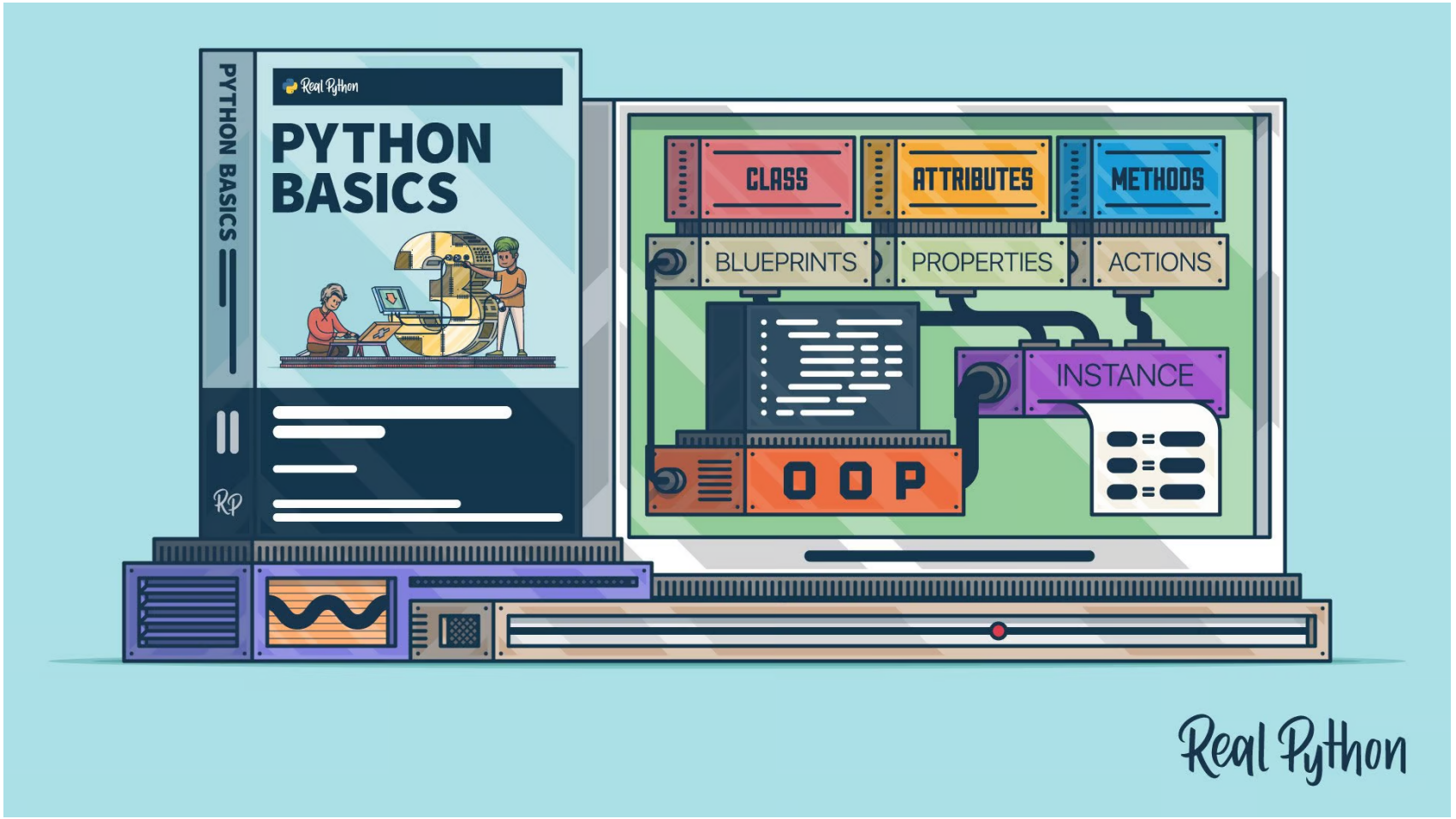
The three steps for each task:

1. 👀 Learn about the exercise
2. 👨‍💻 Code your solution
3. ↔️ Compare your solution

Python Basics Exercises: Object-Oriented Programming

1. Review Exercises
2. Challenges

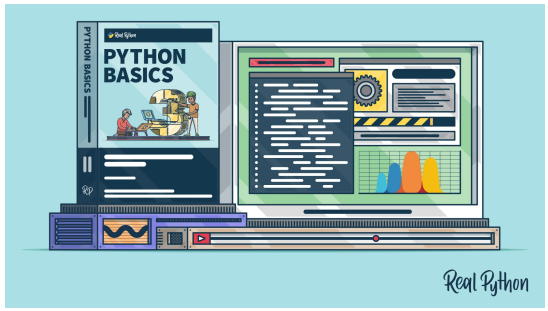
Background - Python Basics: Object-Oriented Programming



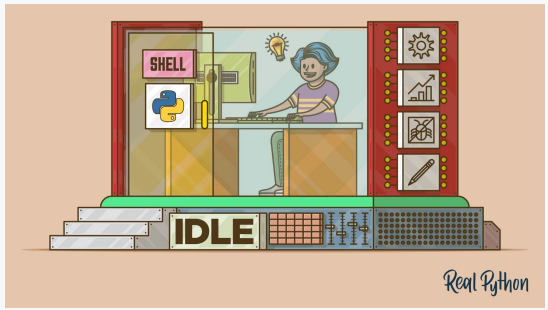
Background - Python Basics: Object-Oriented Programming

- Classes vs instances
- Class definition
- Class attributes
- Instance attributes
- Instance methods
- Special methods (dunder methods)
- F-strings

Background - Using IDLE



Python Basics: Setting Up Python



Getting Started With Python IDLE

Ready to Get Started?

```
class ReviewExercise:
    pass

class Challenge:
    pass
```

🐶 Create the `Dog` Class (Exercise)

Create a `Dog` class that has a class attribute, `species`, that holds the value `"Canis familiaris"`.

Each instance of `Dog` should have a `name` and an `age`.

Also create a readable representation for a `Dog` instance using the `.__str__()` special method.

Finally, create an instance method called `.speak()` that takes a `sound` as an argument and prints a sentence that the dog says the sound.

🐶 Create the `Dog` Class (Exercise)

For example:

```
>>> philo = Dog("Philo", 12)

>>> print(philo)
Philo is 12 years old

>>> philo.speak("Wau")
Philo says Wau
```

💡 Create the `Dog` Class (Solution)

```
class Dog:

    species = "Canis familiaris"

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

    def __str__(self):
        return f"{self.name} is {self.age} years old"

    def speak(self, sound):
        return f"{self.name} says {sound}"
```

🤔 Add Coat Color (Exercise)

Modify the `Dog` class to include a third instance attribute called `coat_color`, which stores the color of the dog's coat as a string. Store your new class in a file and test it out by adding the following code at the bottom of the code:

```
philo = Dog("Philo", 5, "brown")
print(f"{philo.name}'s coat is {philo.coat_color}.")
```

The output of your program should be the following:

```
Philo's coat is brown.
```

Add Coat Color (Solution)

```
class Dog:

    species = "Canis familiaris"

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

    def __str__(self):
        return f"{self.name} is {self.age} years old and their coat is {self.coat_color}"

    def speak(self, sound):
        return f"{self.name} says {sound}"
```

Review Exercises: Recap

- Creating classes
- Defining class attributes
- Defining instance attributes
- Defining instance methods
- Defining special methods
- Working with f-strings

Tips

- Use code comments to help you get organized
- Break exercises into smaller tasks
- Use descriptive variable names
- Test repeatedly to see whether the code does what you expect it to do

Tips

- Use code comments to help you get organized
- Break exercises into smaller tasks
- Use descriptive variable names
- Test repeatedly to see whether the code does what you expect it to do

Next Up: Challenges!

🤔 Challenge: Create a `Car` Class

Create a `Car` class with two instance attributes:

1. `.color`, which stores the name of the car's color as a string
2. `.mileage`, which stores the number of miles on the car as an integer

Then instantiate two `Car` objects:

1. A blue car with 20,000 miles
2. A red car with 30,000 miles

Print out their colors and mileage. Your output should look like this:

```
The blue car has 20,000 miles.
The red car has 30,000 miles.
```

💡 Challenge: Create a Car Class

```
class Car:

    def __init__(self, color, mileage):
        self.color = color
        self.mileage = mileage

    def __str__(self):
        return f"The {self.color} car has {self.mileage:,} miles"


blue_car = Car("blue", 20_000)
red_car = Car("red", 30_000)
print(blue_car)
print(red_car)
```

🤔 Challenge: Let Your Cars Drive

Modify the `Car` class with an instance method called `.drive()`, which takes a number as an argument and adds that number to the `.mileage` attribute.

Test that your solution works by instantiating a car with `0` miles, then call `.drive(100)` and print the `.mileage` attribute to check that it is set to `100`.

Challenge: Let Your Cars Drive

```
class Car:

    def __init__(self, color, mileage):
        self.color = color
        self.mileage = mileage

    def __str__(self):
        return f"The {self.color} car has {self.mileage:,} miles"

    def drive(self, miles):
        self.mileage = self.mileage + miles


blue_car = Car("blue", 0)
blue_car.drive(100)

print(blue_car)
```

Summary and Additional Resources

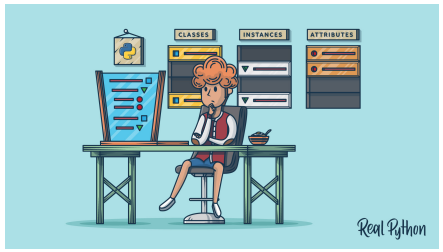
In this course, you **practiced** using:

- Classes
- Class attributes
- Instance attributes
- Instance methods
- Special methods (dunder methods)
- F-strings

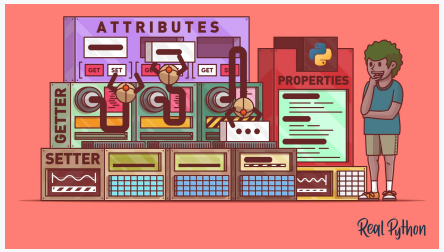
Tips

- Use code comments to help you get organized
- Break exercises into smaller tasks
- Use descriptive variable names
- Test repeatedly to see whether the code does what you expect it to do

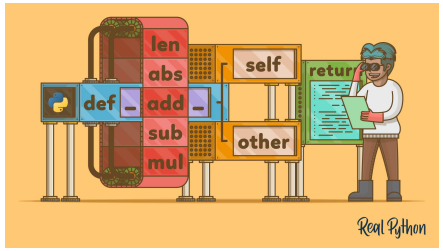
Additional Resources



Object-Oriented Programming (OOP) in Python 3

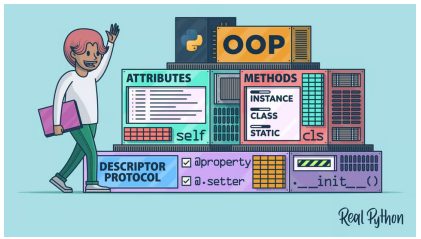


Getters and Setters: Manage Attributes in Python

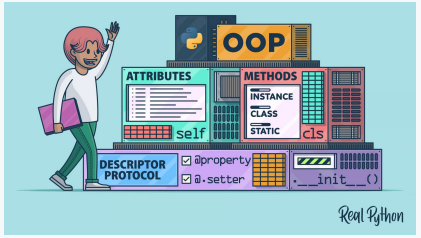


Operator and Function Overloading in Custom Python Classes

Additional Resources



Python Classes: The Power of Object-Oriented Programming



Class Concepts: Object-Oriented Programming in Python

width:230px

Inheritance and Internals: Object-Oriented Programming in Python

width:230px

Design and Guidance: Object-Oriented Programming in Python

Congratulations and Thanks!