PIAS
PHENIKAA INSTITUTE FOR ADVANCED STUDY

PHENIKAA UNIVERSITY
Realizing your potential

PYTHON

PYTHON CLASSES

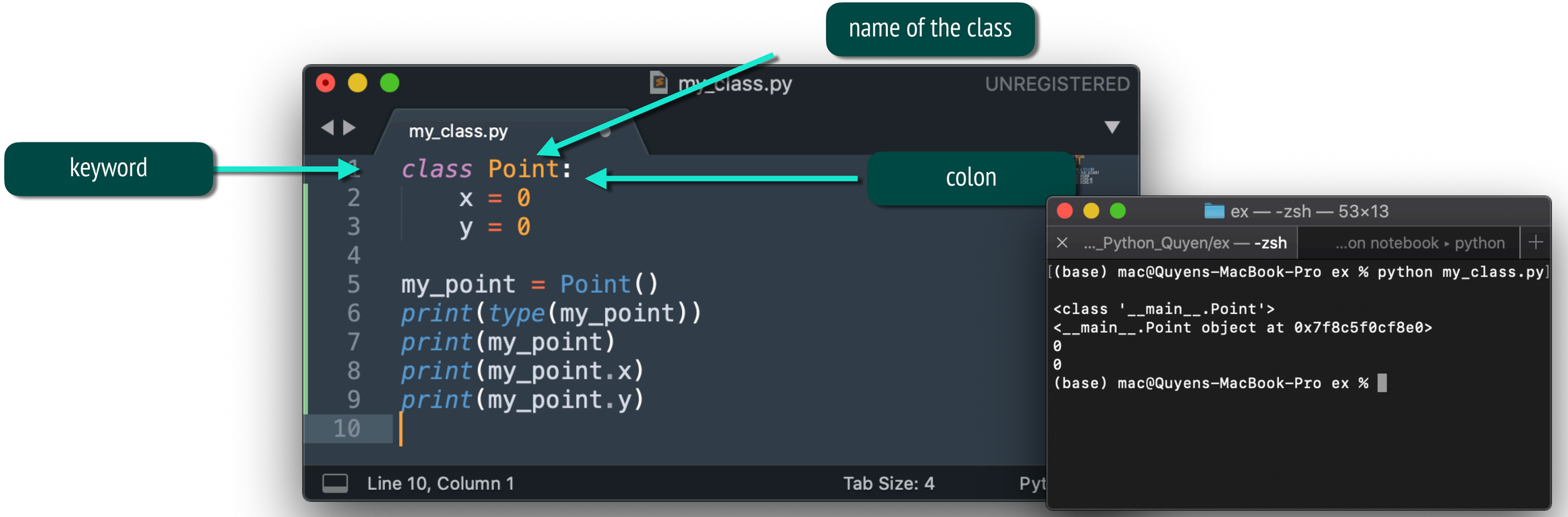
Tien-Lam Pham
Anh-Tuan Nguyen
Phenikaa School of Computing

OUTLINE

- Definition
- Class construction
- Example
- Assignment

DEFINITION

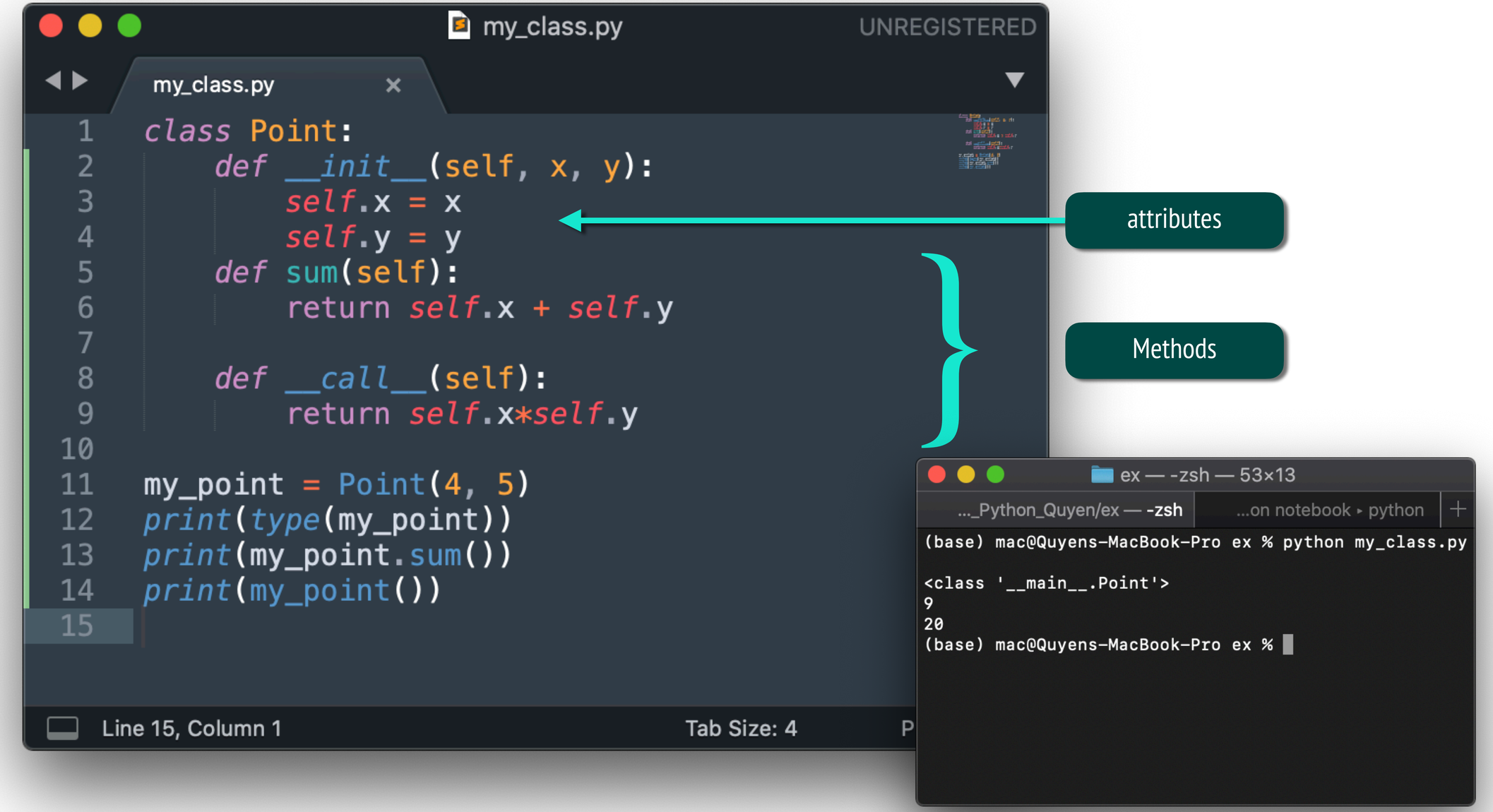
- Classes are the foundation of object-oriented programming
- Classes represent real-world things you want to model in your programs
- A class defines the general behavior that a whole categoy of objects can have and the information that can be associated with those objects



CREATE A CLASS

- The __init__() function is called automatically every time the class is being used to create a new object.
- The self parameter is a reference to the current instance of the class.
- __call__() function: instances behave like functions and can be called like a functions.

CREATE A CLASS



EXAMPLE

class_example.py — UNREGISTERED

my_class.py | class_example.py

```python
class Car():
    """A simple attempt to model a car."""
    def __init__(self, make, model, year):
        """Initialize car attributes."""
        self.make = make
        self.model = model
        self.year = year

        # Fuel capacity and level in gallons.
        self.fuel_capacity = 15
        self.fuel_level = 0

    def fill_tank(self):
        """Fill gas tank to capacity."""
        self.fuel_level = self.fuel_capacity
        print("Fuel tank is full.")

    def drive(self):
        """Simulate driving."""
        print("The car is moving.")

```

Line 21, Column 1 — Spaces: 4 — Python

```python
#Creating an object from a class
my_car = Car("audi", "a4", 2016)
```

```python
#Acessing attribute values
print(my_car.make)
print(my_car.model)
print(my_car.year)
```

```
audi
a4
2016
```

```python
#Calling methods
my_car.fill_tank()
my_car.drive()
```

```
Fuel tank is full.
The car is moving.
```

```python
#Creating multiple objects
my_car = Car('audi', 'a4', 2016)
my_old_car = Car('subaru', 'outback', 2013)
my_truck = Car('toyota', 'tacoma', 2010)
```

EXAMPLE

- Modifying attributes
- You can modify an attribute's value directly, or you can write methods that manage updating values more carefully.

```
#Modifying an attribute directly
my_new_car = Car('audi', 'a4', 2016)
my_new_car.fuel_level = 5
```

```
#Writing a method to update an attribute's value
def update_fuel_level(self, new_level):
    """Update the fuel level."""
    if new_level <= self.fuel_capacity:
        self.fuel_level = new_level
    else:
        print("The tank can't hold that much!")
```

```
#Writing a method to increment an attribute's value
def add_fuel(self, amount):
    """Add fuel to the tank."""
    if (self.fuel_level + amount<= self.fuel_capacity):
        self.fuel_level += amount
        print("Added fuel.")
    else:
        print("The tank won't hold that much.")
```

ASSIGNMENT