Module 01 – Exercise Class

Object-Oriented Programming

TA Thanh Huy

TA Dương Đình Thắng

MSc. Nguyen Quoc Thai

Objectives

OOP

- ❖ Class
- ❖ Object
- ❖ Encapsulation
- ❖ Abstraction
- ❖ Inheritance
- ❖ Polymorphism

Exercise

- ❖ nn.Module Pytorch
- ❖ Sigmoid
- ❖ User management
- ❖ Stack
- ❖ Queue

Outline

SECTION 1
OOP Review

SECTION 2
OOP in Pytorch

SECTION 3
Characteristics of OOP

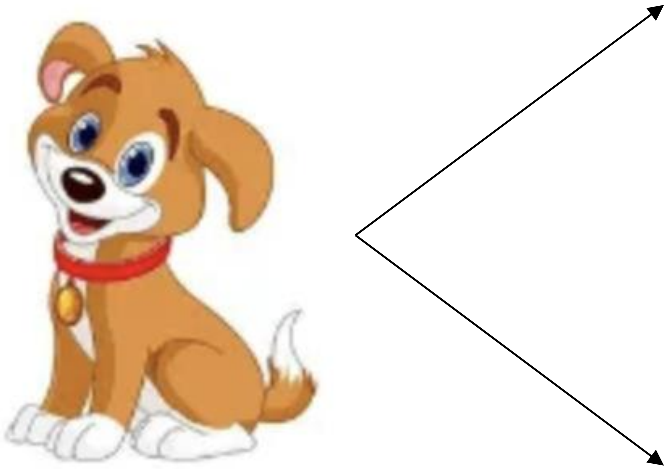
SECTION 4
Stack

SECTION 4
Queue

Review

Class and Object

- An **object** is any entity that has **attributes** and **behaviors**
- A dog is an object



Attributes: name, size, age, color

Behaviors: eat, sleep, sit, run

Review

Class and Object

- A class is a template for objects

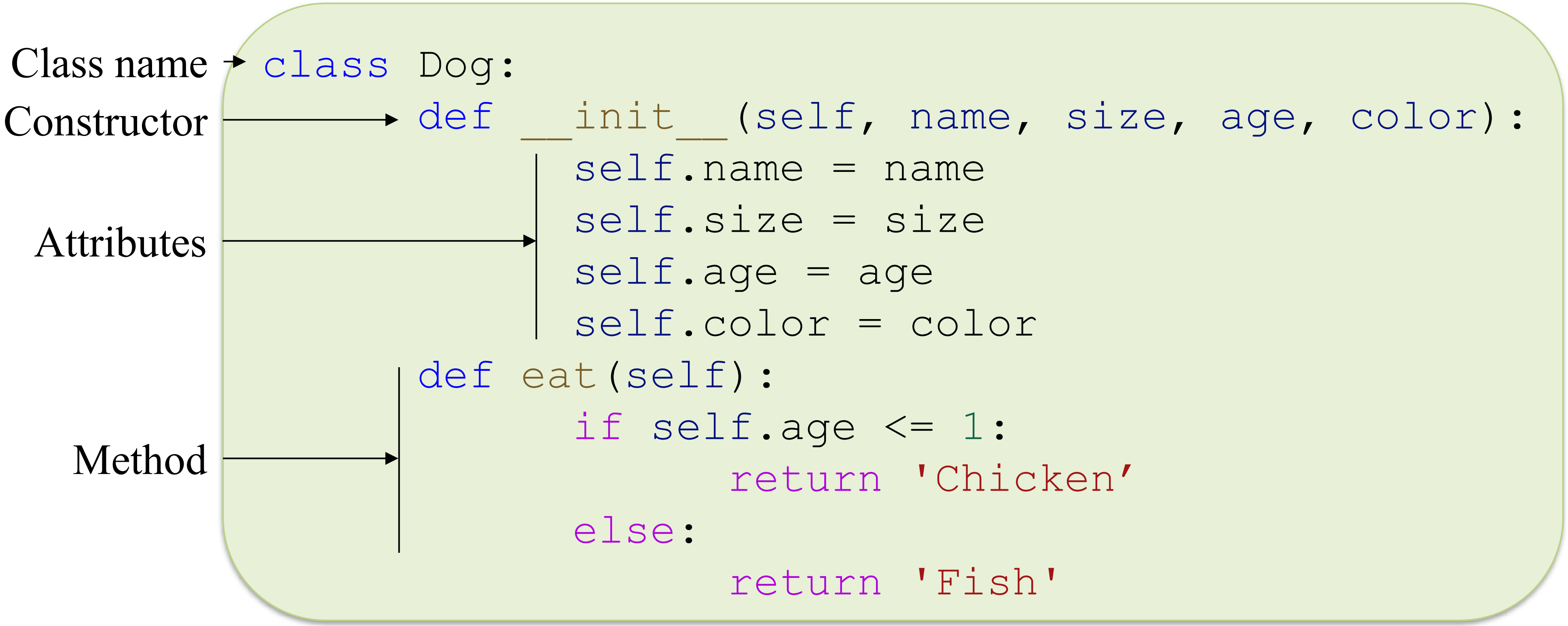
Attributes: name, size, age, color

name	Chow Chow
size	Small
age	2
color	Brown

Behaviors: eat, sleep, sit, run

Review

Class and Object



Class name

Constructor

Attributes

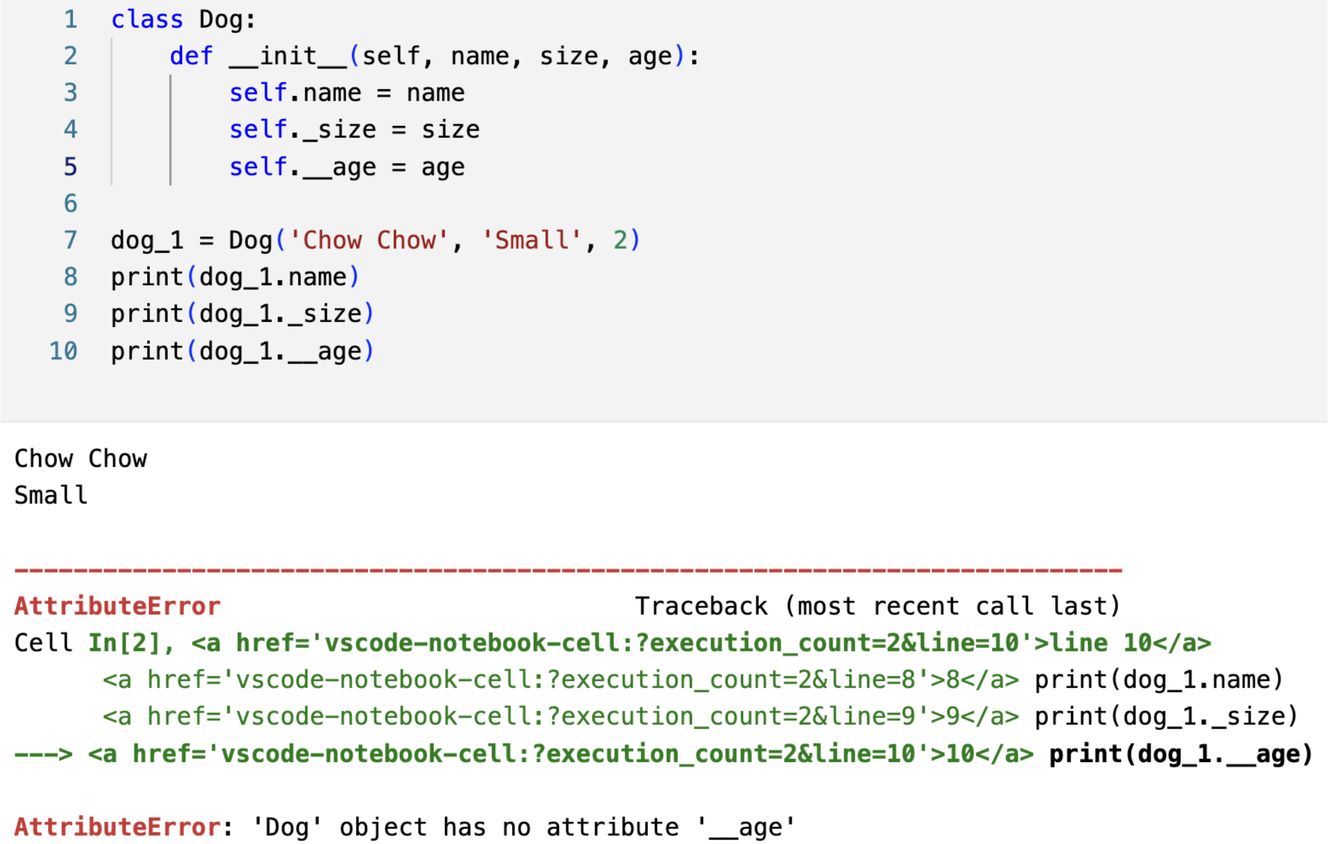
Method

```
class Dog:
    def __init__(self, name, size, age, color):
        self.name = name
        self.size = size
        self.age = age
        self.color = color
    def eat(self):
        if self.age <= 1:
            return 'Chicken’
        else:
            return 'Fish'
```

Review

Encapsulation

- Information hiding and limit access
- Access modifiers: Public, Protected, Private



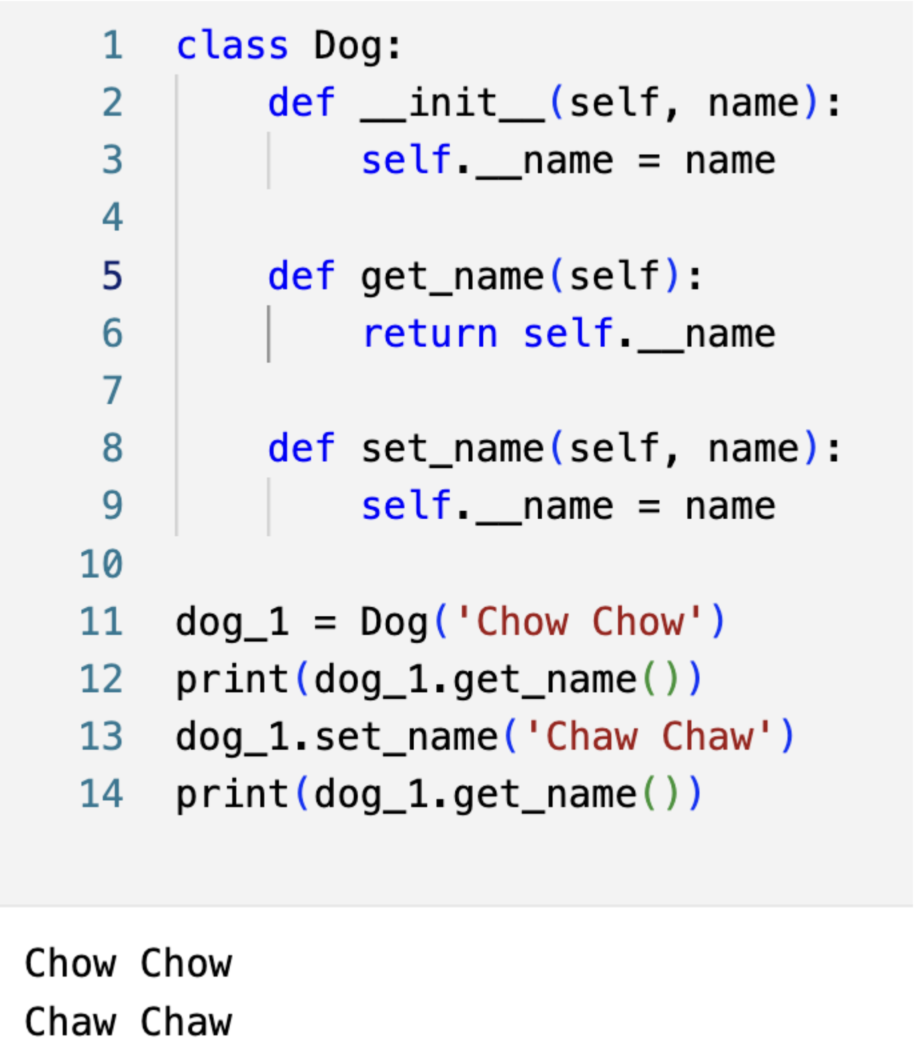
```python
class Dog:
    def __init__(self, name, size, age):
        self.name = name
        self._size = size
        self.__age = age

dog_1 = Dog('Chow Chow', 'Small', 2)
print(dog_1.name)
print(dog_1._size)
print(dog_1.__age)
```

```
Chow Chow
Small

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[2], <a href='vscode-notebook-cell:?execution_count=2&line=10'>line 10</a>
      <a href='vscode-notebook-cell:?execution_count=2&line=8'>8</a> print(dog_1.name)
      <a href='vscode-notebook-cell:?execution_count=2&line=9'>9</a> print(dog_1._size)
---> <a href='vscode-notebook-cell:?execution_count=2&line=10'>10</a> print(dog_1.__age)

AttributeError: 'Dog' object has no attribute '__age'
```

Review

Encapsulation

- Information hiding and limit access
- Access modifiers: Public, Protected, Private
- Ensure data encapsulation: getter, setter



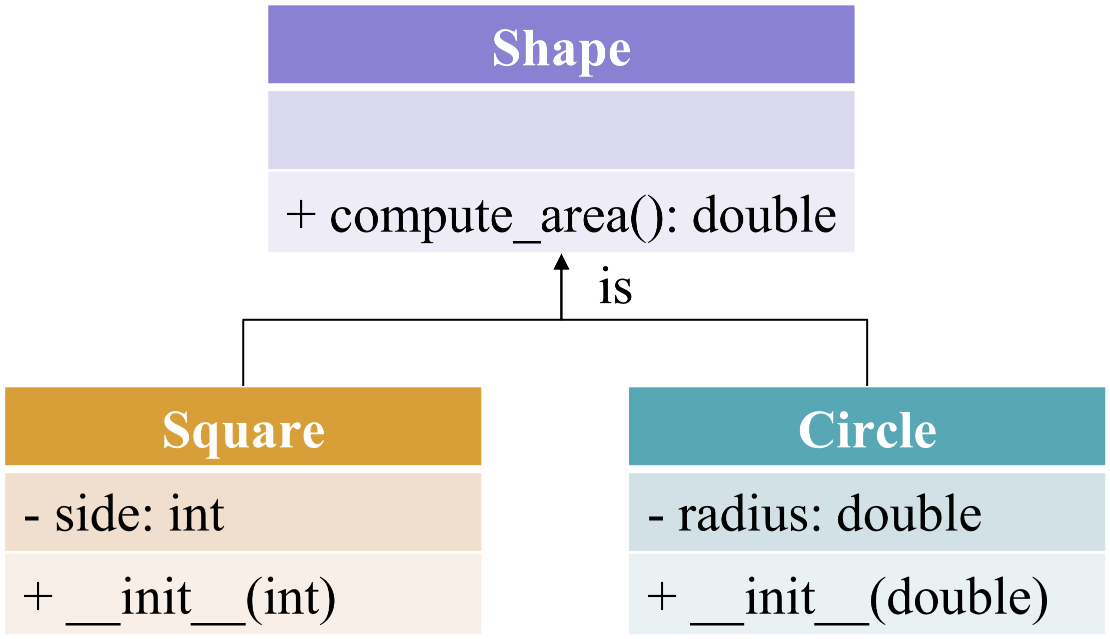
```python
class Dog:
    def __init__(self, name):
        self.__name = name

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name

dog_1 = Dog('Chow Chow')
print(dog_1.get_name())
dog_1.set_name('Chaw Chaw')
print(dog_1.get_name())
```

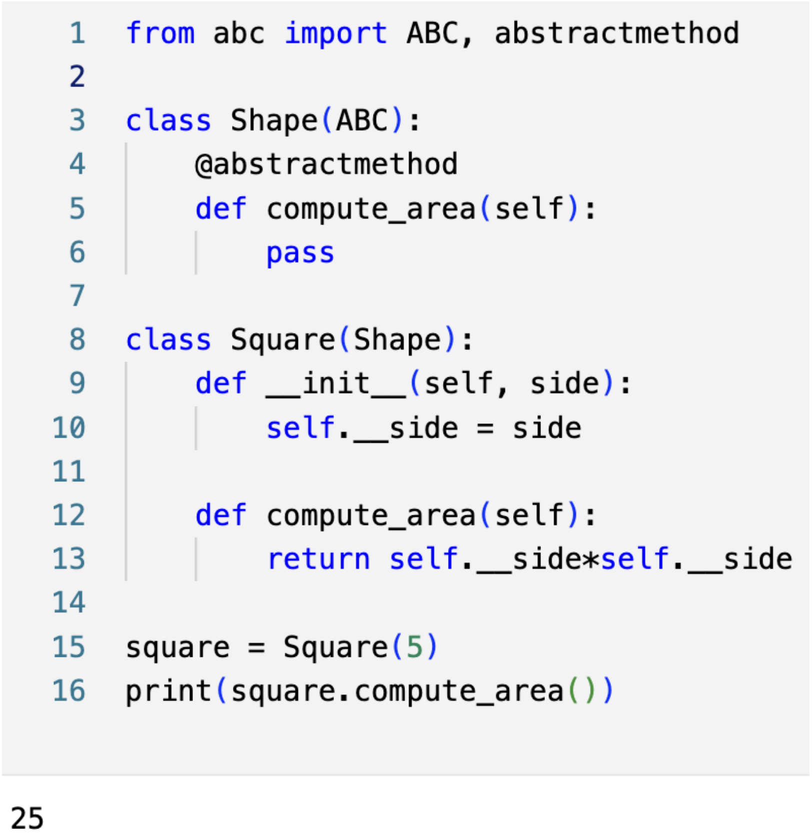
```
Chow Chow
Chaw Chaw
```

Review

! Abstraction

- ➢ Focus only on relevant data of an object
- ➢ Hide the background details and emphasizes the essential data points





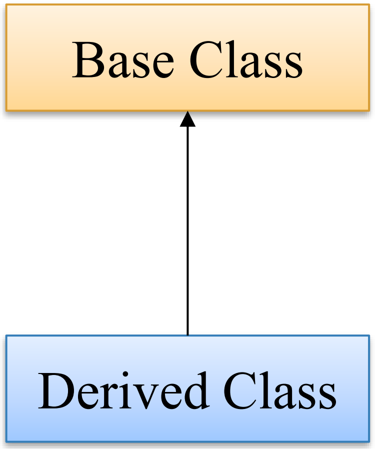
```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def compute_area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.__side = side

    def compute_area(self):
        return self.__side*self.__side

square = Square(5)
print(square.compute_area())
```

```
25
```

Review

Inheritance

- Inheritance is a way of creating a new class for using details of an existing class without modifying it

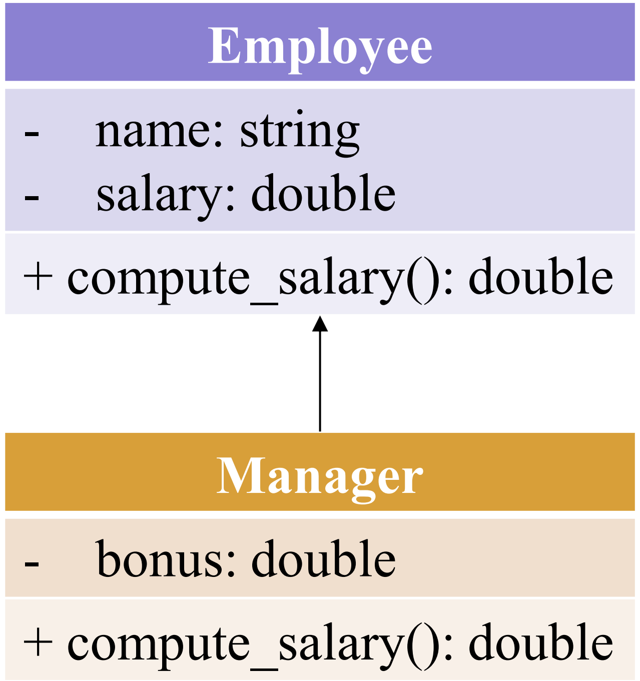


Base class (Parent): the class which is inherited from another class

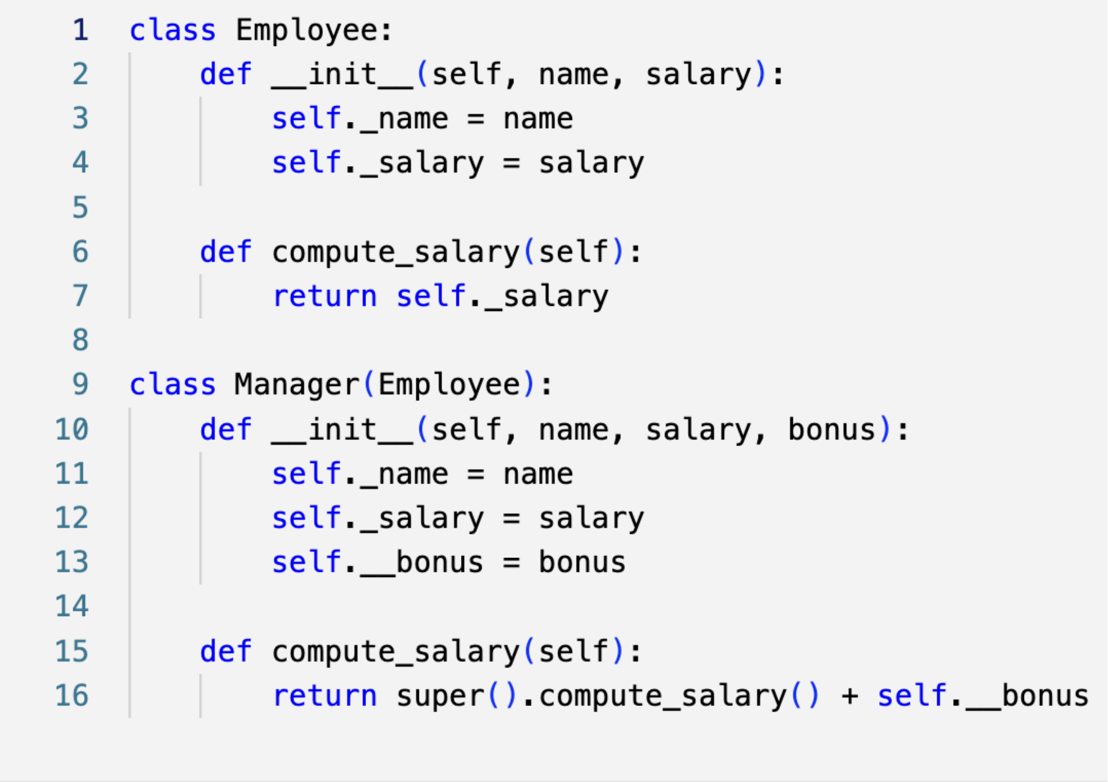
Derived class (Child): the class inherits from another class

Review

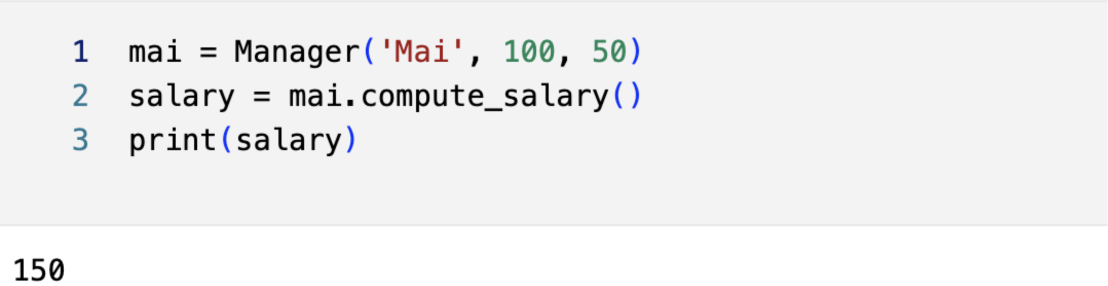
Inheritance


Employee
- name: string
- salary: double
+ compute_salary(): double

Manager
- bonus: double
+ compute_salary(): double



```python
class Employee:
    def __init__(self, name, salary):
        self._name = name
        self._salary = salary

    def compute_salary(self):
        return self._salary

class Manager(Employee):
    def __init__(self, name, salary, bonus):
        self._name = name
        self._salary = salary
        self.__bonus = bonus

    def compute_salary(self):
        return super().compute_salary() + self.__bonus
```

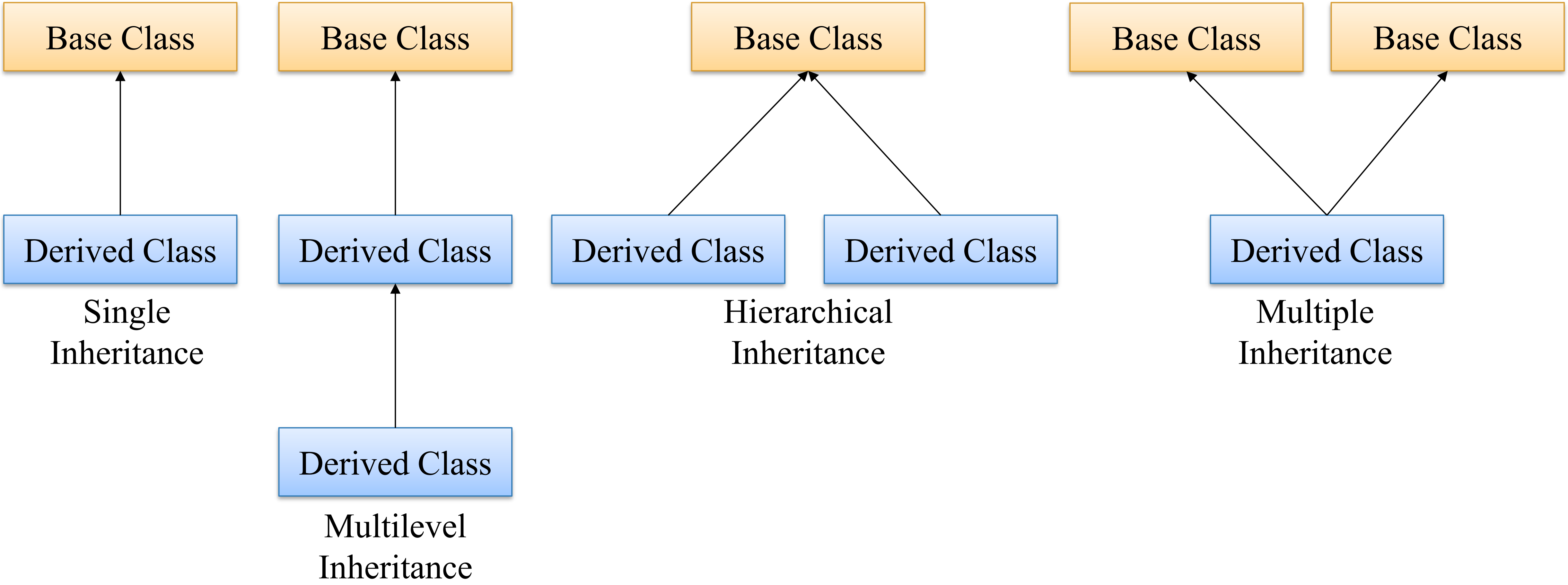


```python
mai = Manager('Mai', 100, 50)
salary = mai.compute_salary()
print(salary)
```

```
150
```

Review

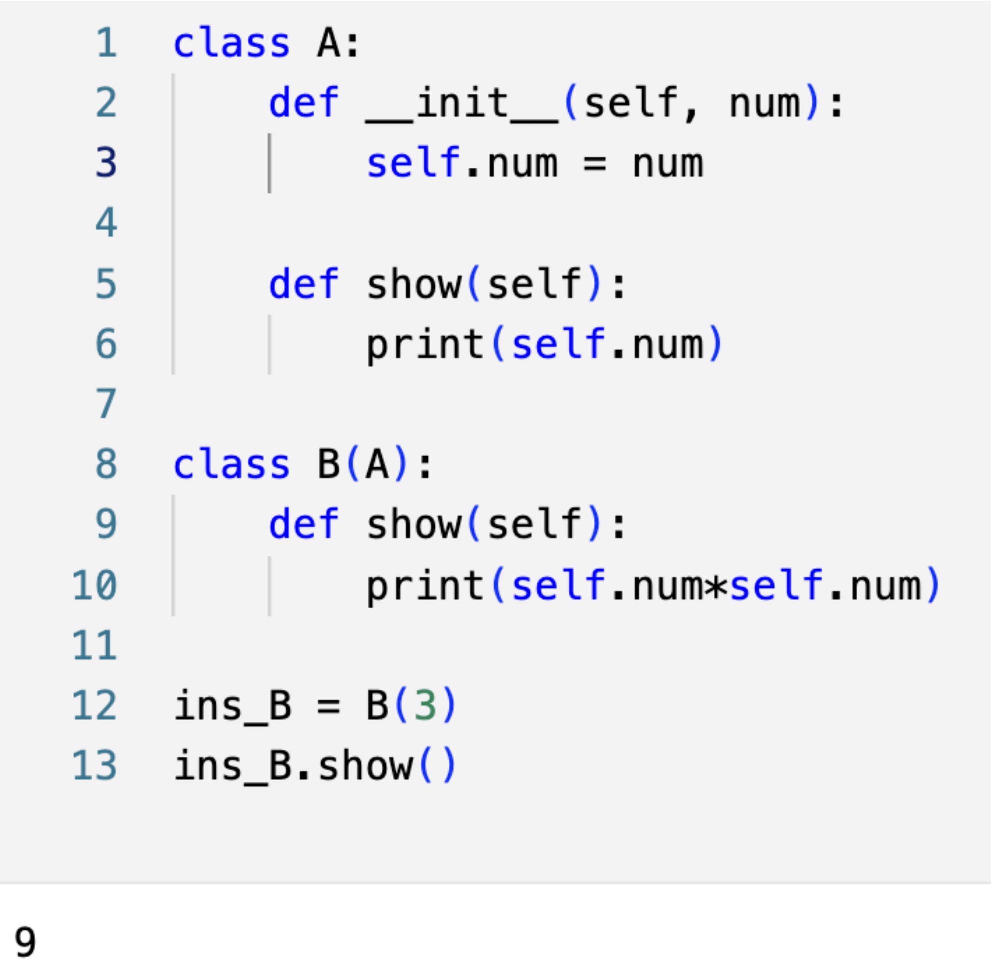
Inheritance



Review

Polymorphism

- Use a single type entity (method, operator or object) to represent different types in different scenarios
- Method overriding, method overloading (not support in Python)



```python
class A:
    def __init__(self, num):
        self.num = num

    def show(self):
        print(self.num)

class B(A):
    def show(self):
        print(self.num*self.num)

ins_B = B(3)
ins_B.show()
```

```
9
```

Outline

SECTION 1

OOP Review

SECTION 2

OOP in Pytorch

SECTION 3

Characteristics of OOP

SECTION 4

Stack

SECTION 4

Queue

OOP in PyTorch

Solution

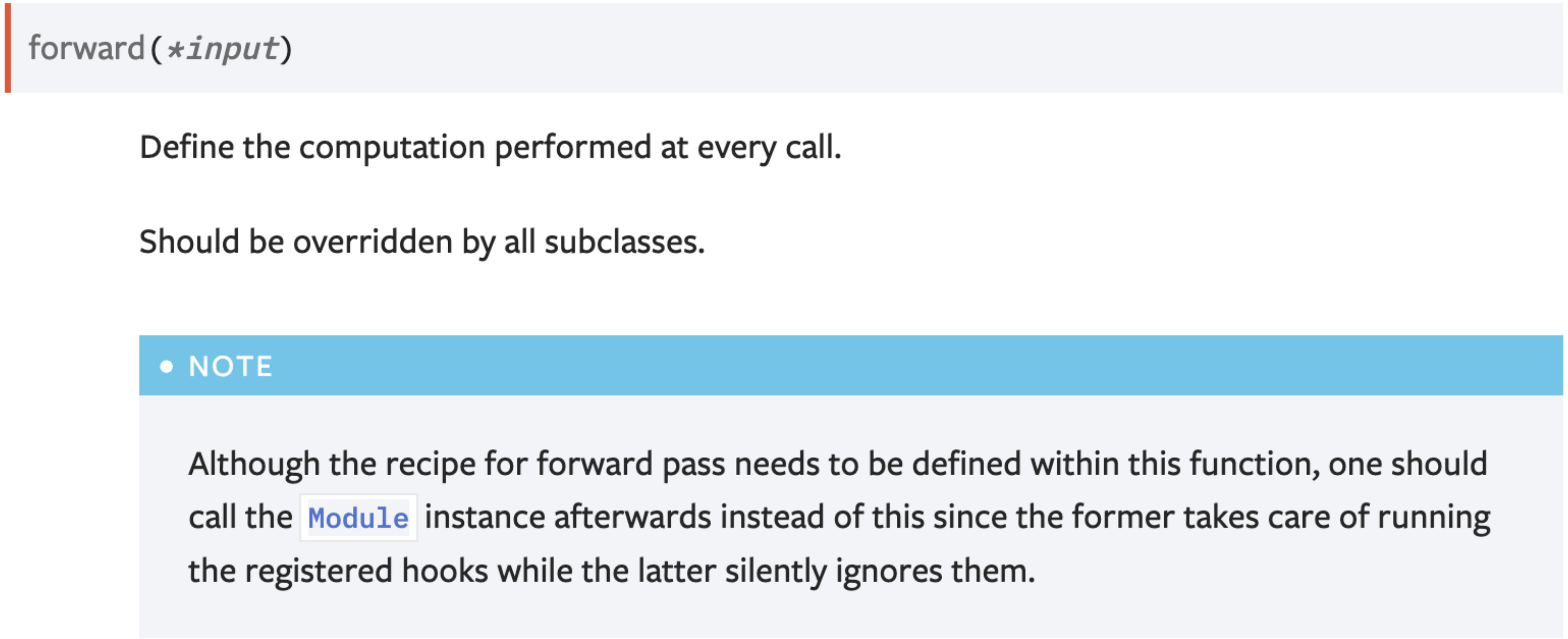
Problem: Dựa vào class torch.nn.Module, xây dựng các class để tính hàm sigmoid như sau:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

OOP in PyTorch

Torch.nn.Module

- Base class for all neural network modules, activation functions,…
- Forward() method



`forward(*input)`

Define the computation performed at every call.

Should be overridden by all subclasses.

● NOTE

Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

OOP in PyTorch

Sigmoid

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

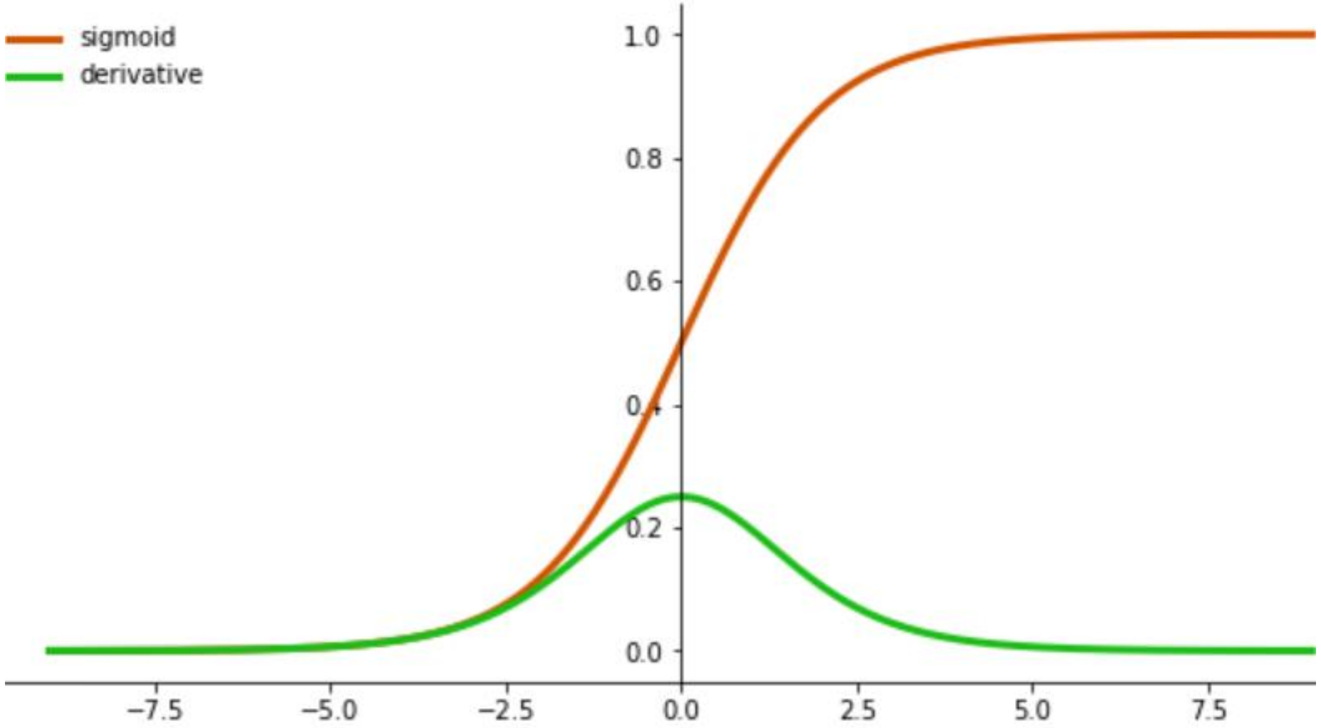
data =

1	5	-4	3	-2

data_a = sigmoid(data)

data_a =

0.731	0.993	0.017	0.95	0.119



OOP in PyTorch

Sigmoid

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

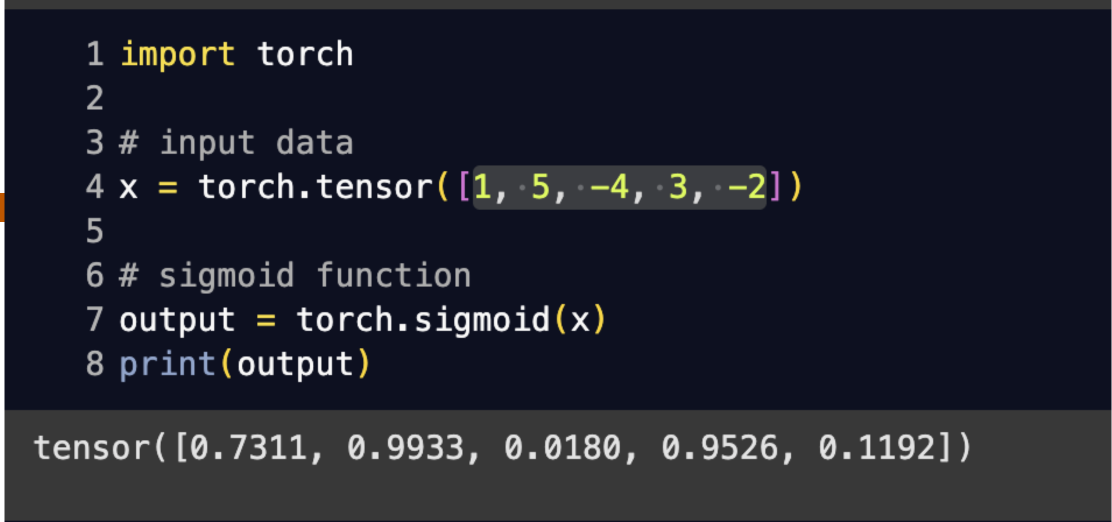
data =

1	5	-4	3	-2

data_a = sigmoid(data)

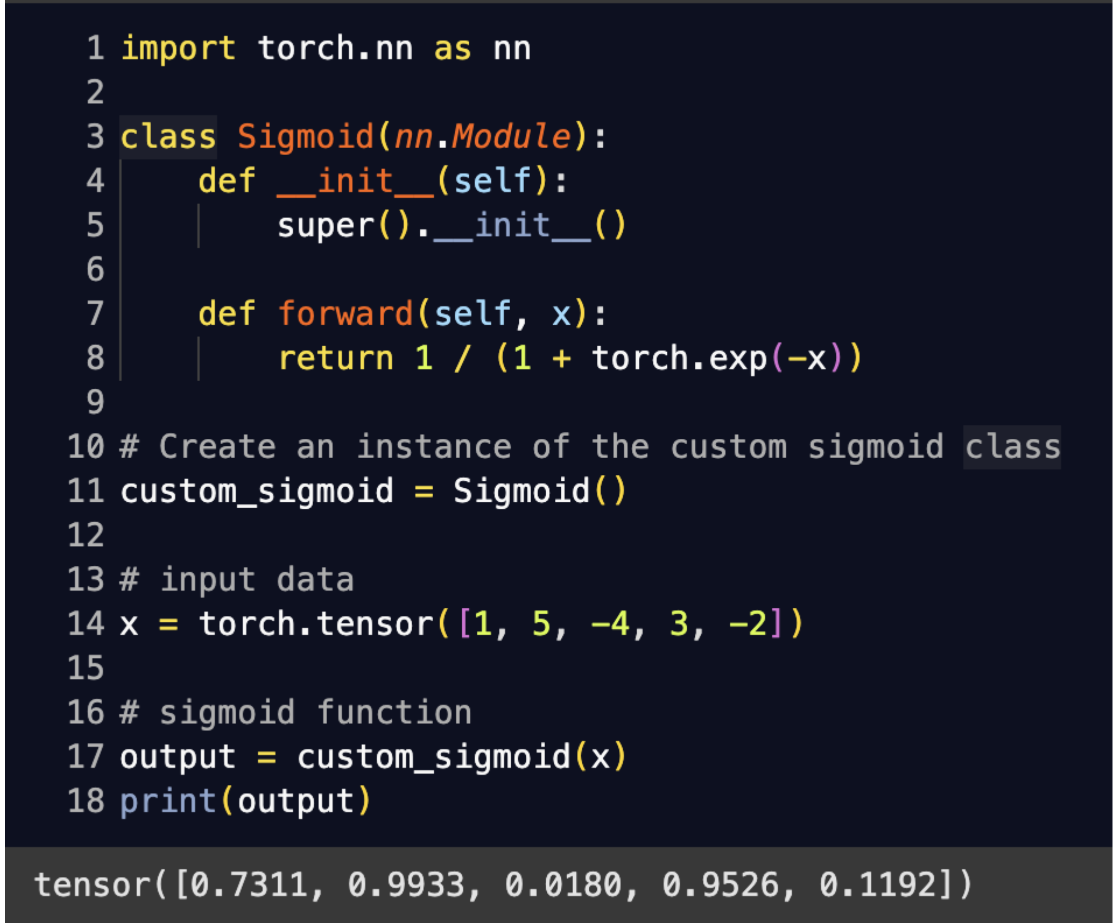
data_a =

0.731	0.993	0.017	0.95	0.119



```python
import torch

# input data
x = torch.tensor([1, 5, -4, 3, -2])

# sigmoid function
output = torch.sigmoid(x)
print(output)
```

```
tensor([0.7311, 0.9933, 0.0180, 0.9526, 0.1192])
```



```python
import torch.nn as nn

class Sigmoid(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, x):
        return 1 / (1 + torch.exp(-x))

# Create an instance of the custom sigmoid class
custom_sigmoid = Sigmoid()

# input data
x = torch.tensor([1, 5, -4, 3, -2])

# sigmoid function
output = custom_sigmoid(x)
print(output)
```

```
tensor([0.7311, 0.9933, 0.0180, 0.9526, 0.1192])
```

Outline

SECTION 1
OOP Review

SECTION 2
OOP in Pytorch

SECTION 3
Characteristics of OOP

SECTION 4
Stack

SECTION 4
Queue

Characteristics of OOP

! Description

Problem: Một Ward gồm có name (string) và danh sách của mọi người trong Ward. Một người Person có thể là Student, Doctor, hoặc Teacher. Một Student gồm có name, yob (int) (năm sinh), và grade (string). Một Teacher gồm có name, yob, và subject (string). Một Doctor gồm có name, yob, và specialist (string). Lưu ý cần sử dụng một danh sách để chứa danh sách của mọi người trong Ward.

Characteristics of OOP

Description

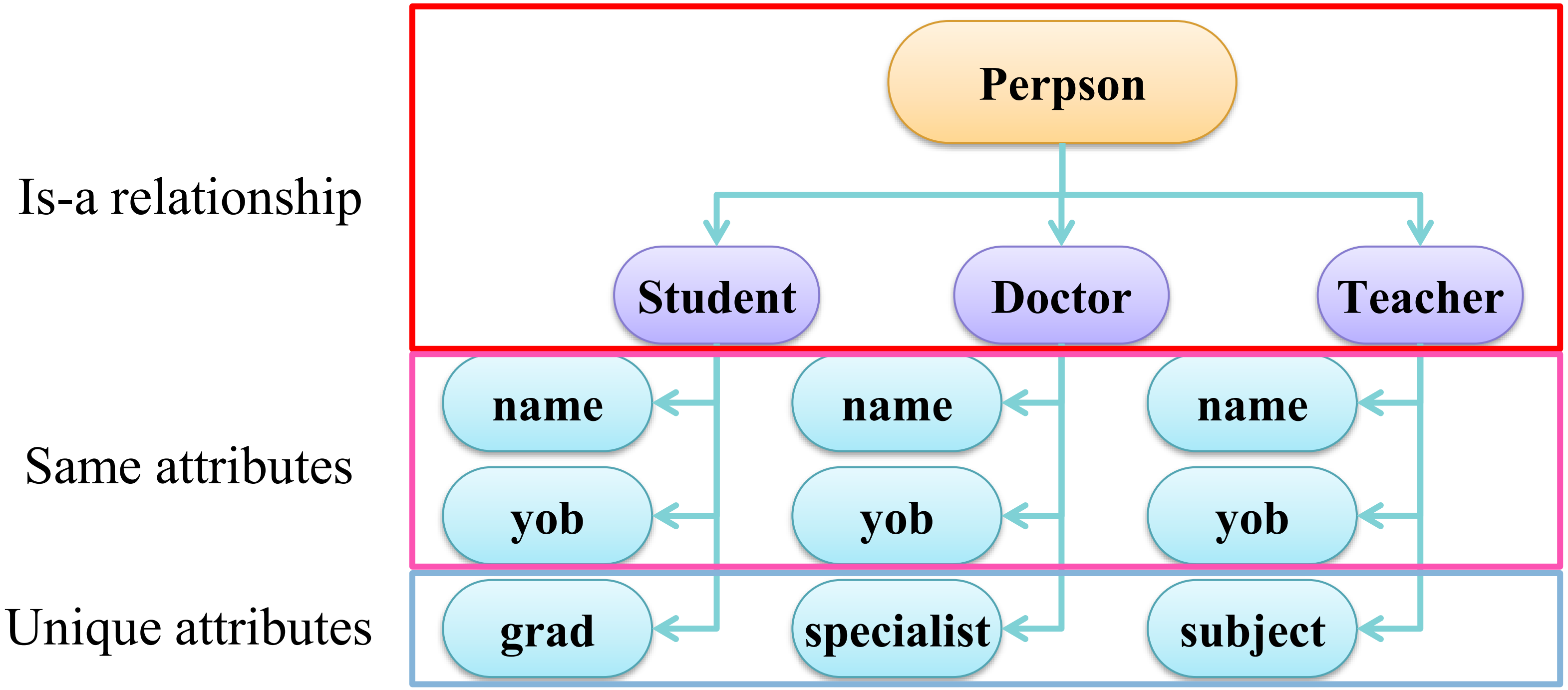
Problem: Một Ward gồm có name (string) và danh sách của mọi người trong Ward. Một người Person có thể là Student, Doctor, hoặc Teacher. Một Student gồm có name, yob (int) (năm sinh), và grade (string). Một Teacher gồm có name, yob, và subject (string). Một Doctor gồm có name, yob, và specialist (string). Lưu ý cần sử dụng một danh sách để chứa danh sách của mọi người trong Ward.

Ward
- name: string - list_people(): list

Characteristics of OOP

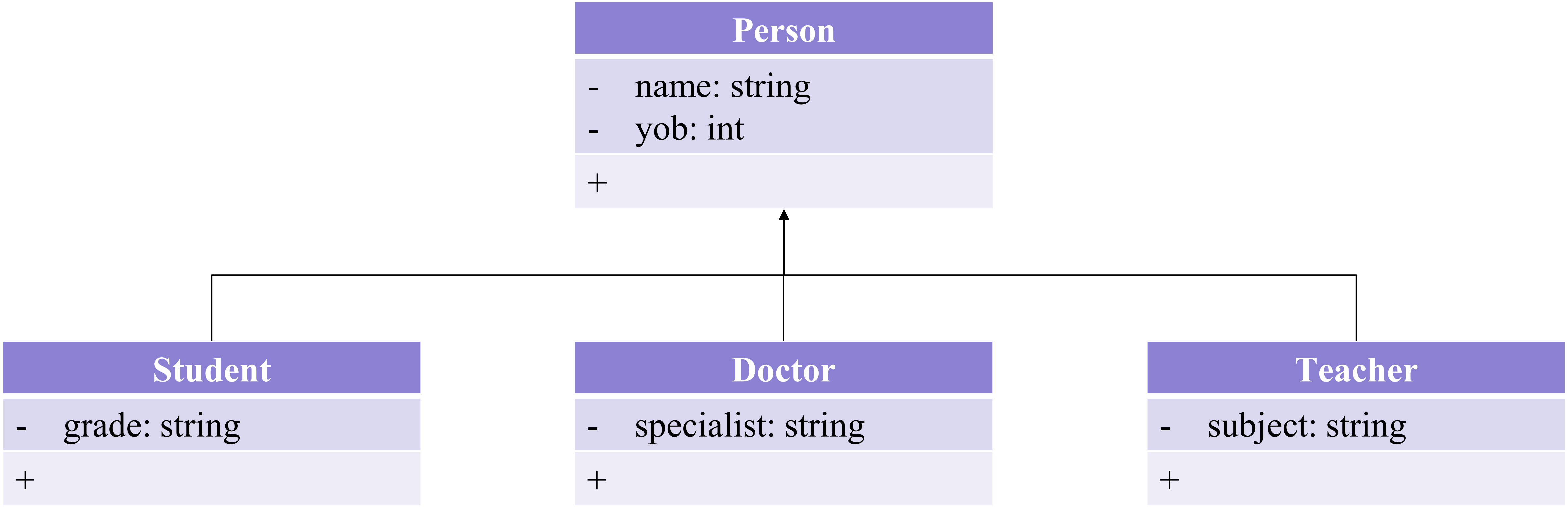
Description

Problem: Một Ward gồm có name (string) và danh sách của mọi người trong Ward. Một người Person có thể là Student, Doctor, hoặc Teacher. Một Student gồm có name, yob (int) (năm sinh), và grade (string). Một Teacher gồm có name, yob, và subject (string). Một Doctor gồm có name, yob, và specialist (string). Lưu ý cần sử dụng một danh sách để chứa danh sách của mọi người trong Ward.



Characteristics of OOP

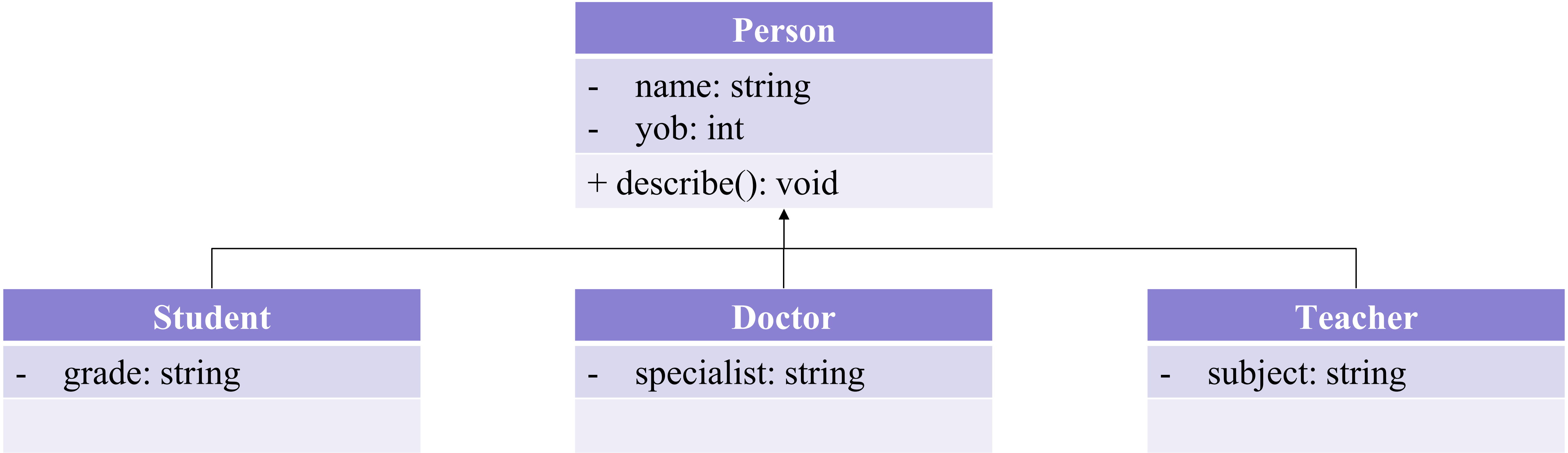
Description



Characteristics of OOP

Description

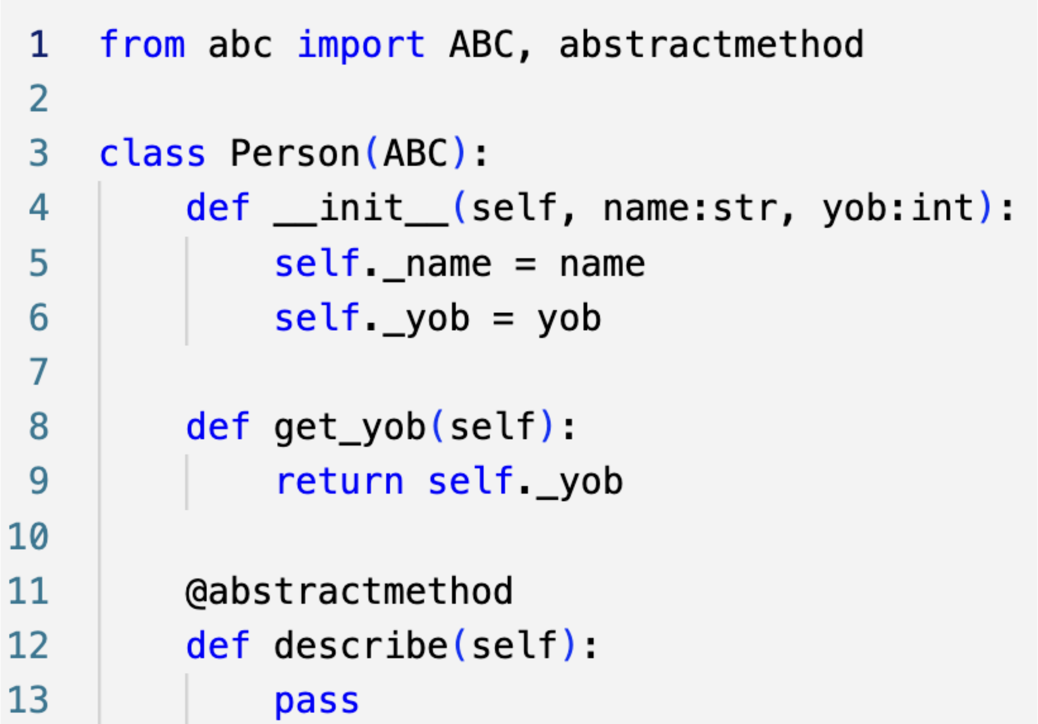
(a): Thực hiện các class student, doctor, và teacher theo mô tả trên. Thực hiện describe() method để print ra tất cả thông tin của các objects.



Characteristics of OOP

Description

(a): Thực hiện các class student, doctor, và teacher theo mô tả trên. Thực hiện describe() method để print ra tất cả thông tin của các objects.

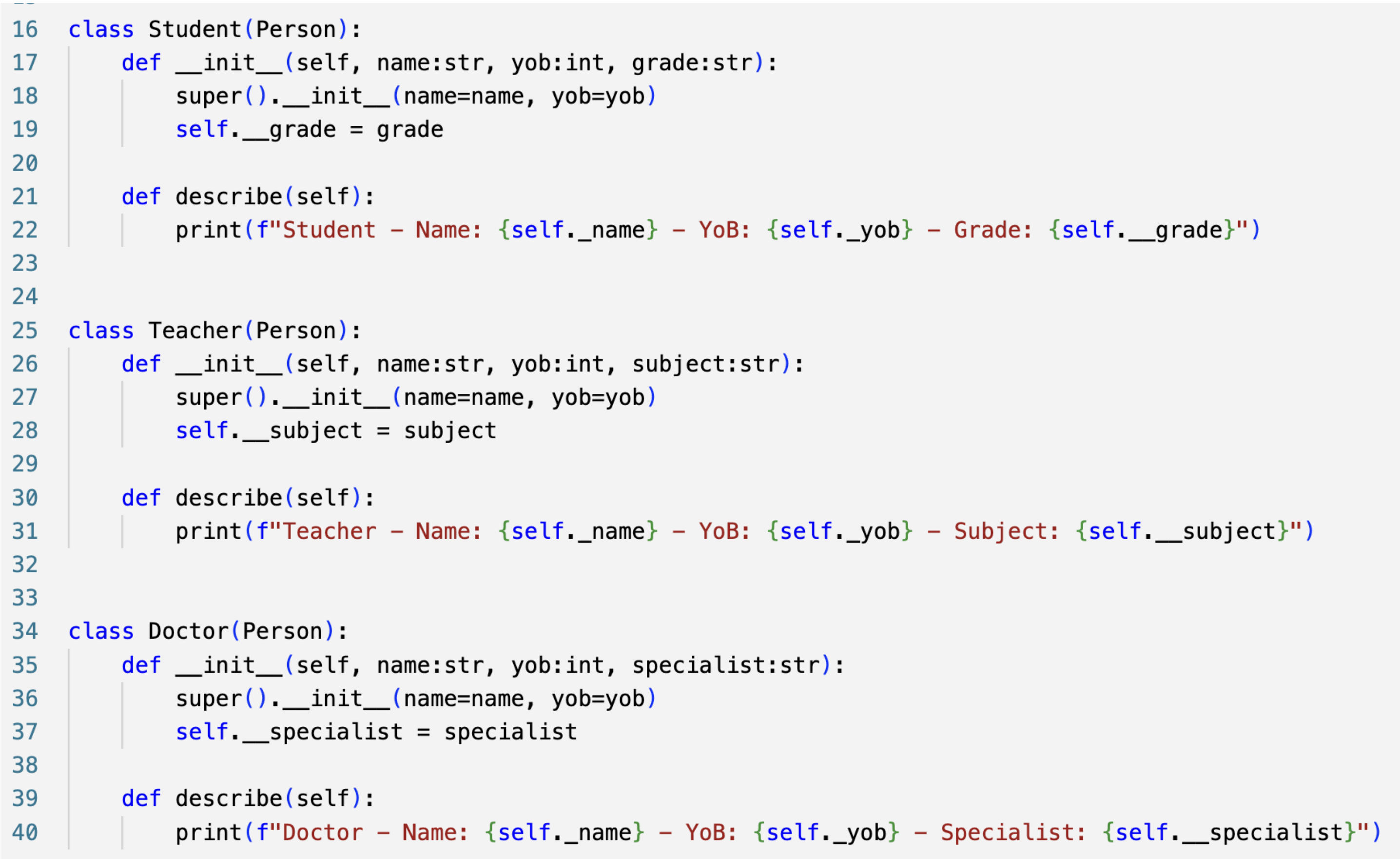


```python
from abc import ABC, abstractmethod

class Person(ABC):
    def __init__(self, name:str, yob:int):
        self._name = name
        self._yob = yob

    def get_yob(self):
        return self._yob

    @abstractmethod
    def describe(self):
        pass
```

Characteristics of OOP

Description

(a): Thực hiện các class student, doctor, và teacher theo mô tả trên. Thực hiện describe() method để print ra tất cả thông tin của các objects.

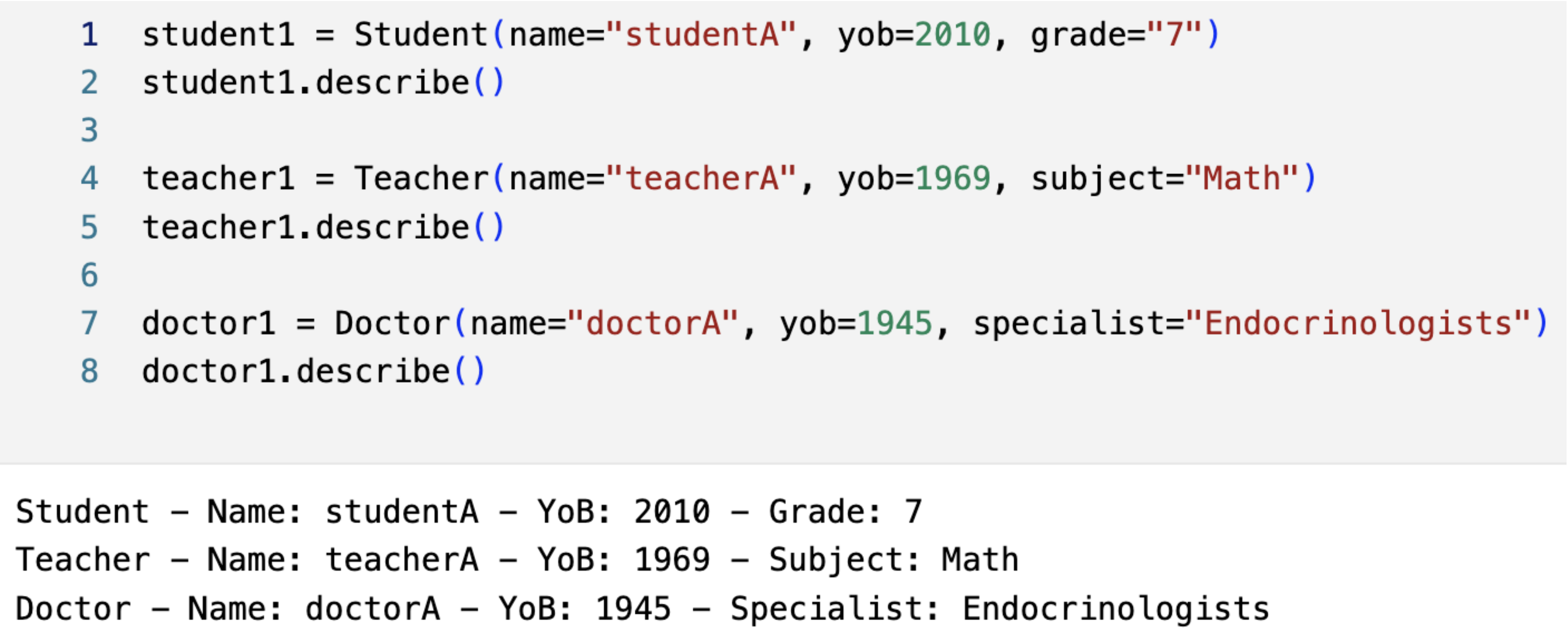


```python
class Student(Person):
    def __init__(self, name:str, yob:int, grade:str):
        super().__init__(name=name, yob=yob)
        self.__grade = grade

    def describe(self):
        print(f"Student - Name: {self._name} - YoB: {self._yob} - Grade: {self.__grade}")


class Teacher(Person):
    def __init__(self, name:str, yob:int, subject:str):
        super().__init__(name=name, yob=yob)
        self.__subject = subject

    def describe(self):
        print(f"Teacher - Name: {self._name} - YoB: {self._yob} - Subject: {self.__subject}")


class Doctor(Person):
    def __init__(self, name:str, yob:int, specialist:str):
        super().__init__(name=name, yob=yob)
        self.__specialist = specialist

    def describe(self):
        print(f"Doctor - Name: {self._name} - YoB: {self._yob} - Specialist: {self.__specialist}")
```

Characteristics of OOP

Description

(a): Thực hiện các class student, doctor, và teacher theo mô tả trên. Thực hiện describe() method để print ra tất cả thông tin của các objects.



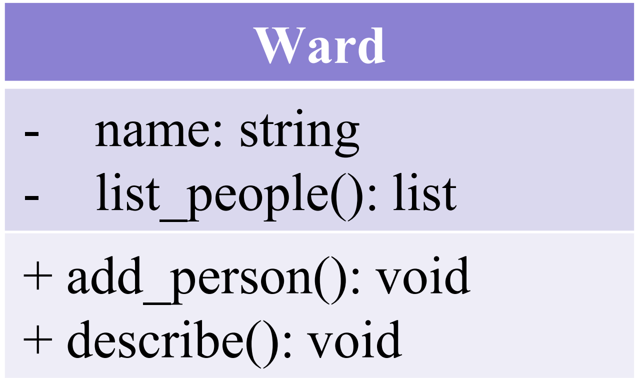
```python
student1 = Student(name="studentA", yob=2010, grade="7")
student1.describe()

teacher1 = Teacher(name="teacherA", yob=1969, subject="Math")
teacher1.describe()

doctor1 = Doctor(name="doctorA", yob=1945, specialist="Endocrinologists")
doctor1.describe()
```

```
Student - Name: studentA - YoB: 2010 - Grade: 7
Teacher - Name: teacherA - YoB: 1969 - Subject: Math
Doctor - Name: doctorA - YoB: 1945 - Specialist: Endocrinologists
```

Characteristics of OOP

Description

(b): add_person(person) method.



Ward

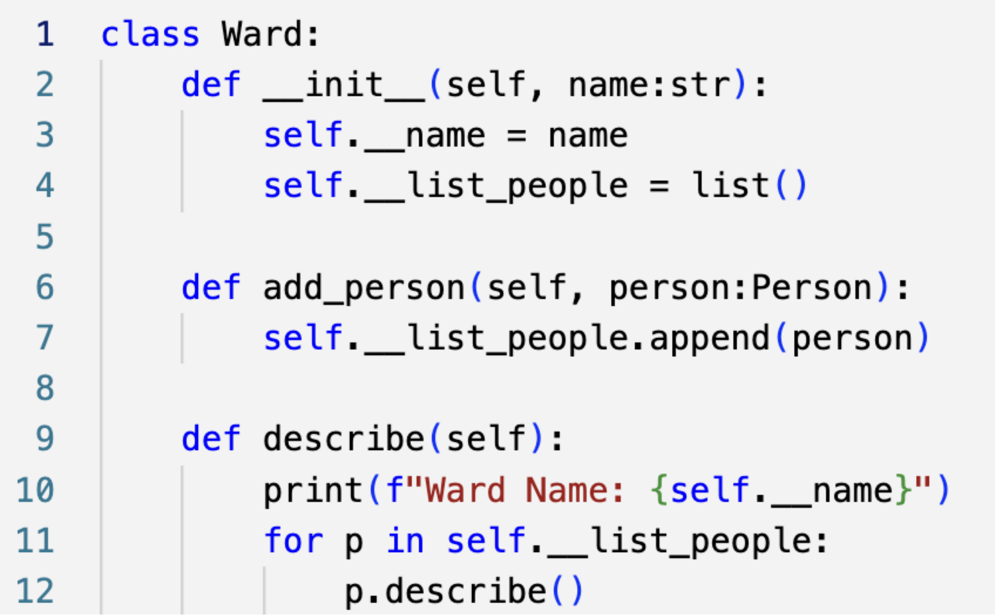
- name: string
- list_people(): list

+ add_person(): void
+ describe(): void

Characteristics of OOP

Description

(b): add_person(person) method.

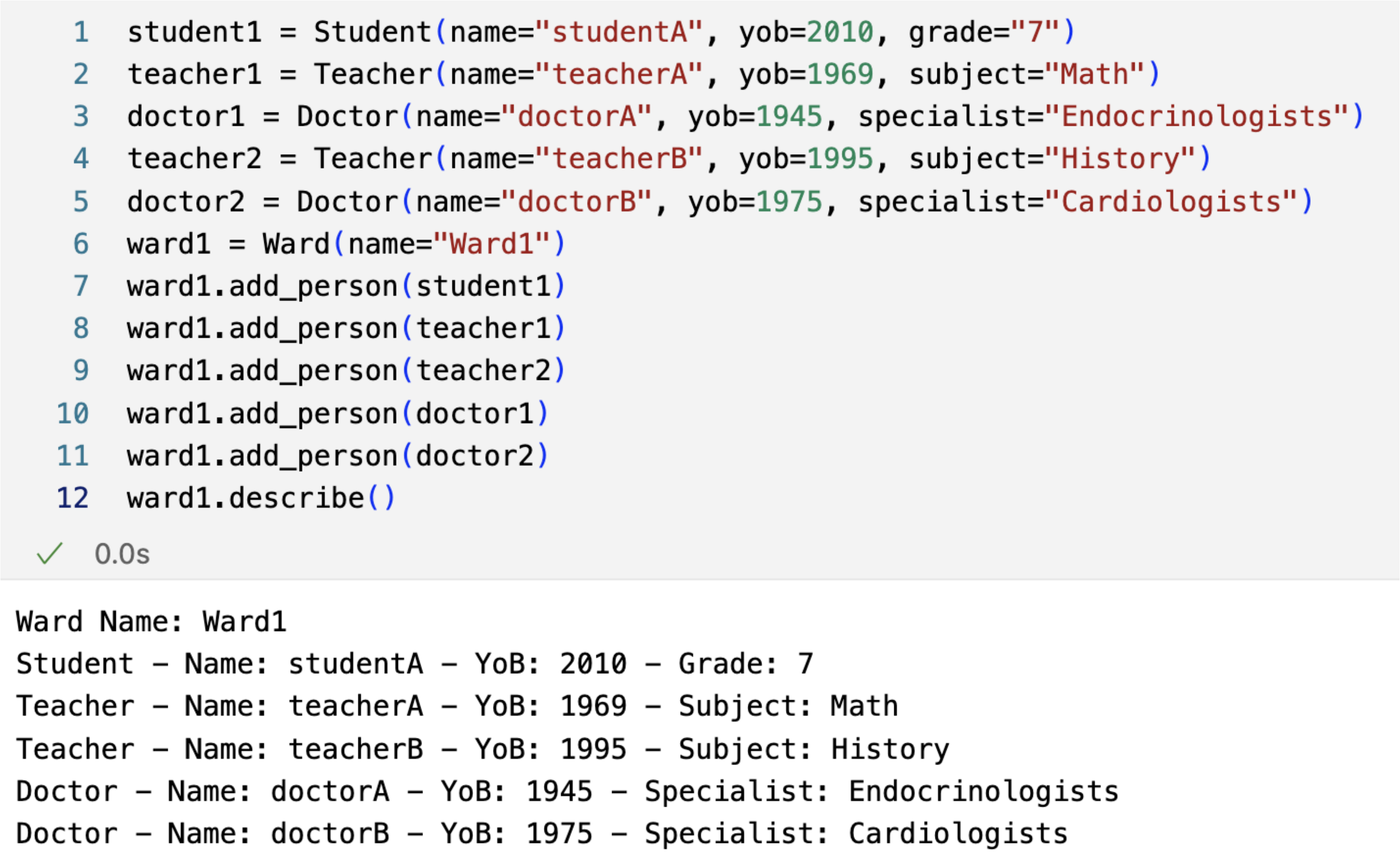


```python
class Ward:
    def __init__(self, name:str):
        self.__name = name
        self.__list_people = list()

    def add_person(self, person:Person):
        self.__list_people.append(person)

    def describe(self):
        print(f"Ward Name: {self.__name}")
        for p in self.__list_people:
            p.describe()
```

Characteristics of OOP

Description

(b): add_person(person) method.



```python
student1 = Student(name="studentA", yob=2010, grade="7")
teacher1 = Teacher(name="teacherA", yob=1969, subject="Math")
doctor1 = Doctor(name="doctorA", yob=1945, specialist="Endocrinologists")
teacher2 = Teacher(name="teacherB", yob=1995, subject="History")
doctor2 = Doctor(name="doctorB", yob=1975, specialist="Cardiologists")
ward1 = Ward(name="Ward1")
ward1.add_person(student1)
ward1.add_person(teacher1)
ward1.add_person(teacher2)
ward1.add_person(doctor1)
ward1.add_person(doctor2)
ward1.describe()
```

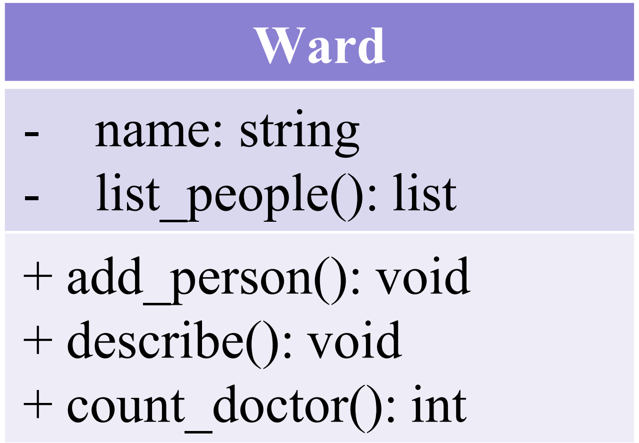
✓ 0.0s

```
Ward Name: Ward1
Student - Name: studentA - YoB: 2010 - Grade: 7
Teacher - Name: teacherA - YoB: 1969 - Subject: Math
Teacher - Name: teacherB - YoB: 1995 - Subject: History
Doctor - Name: doctorA - YoB: 1945 - Specialist: Endocrinologists
Doctor - Name: doctorB - YoB: 1975 - Specialist: Cardiologists
```

Characteristics of OOP

Description

(c): count_doctor().



Ward

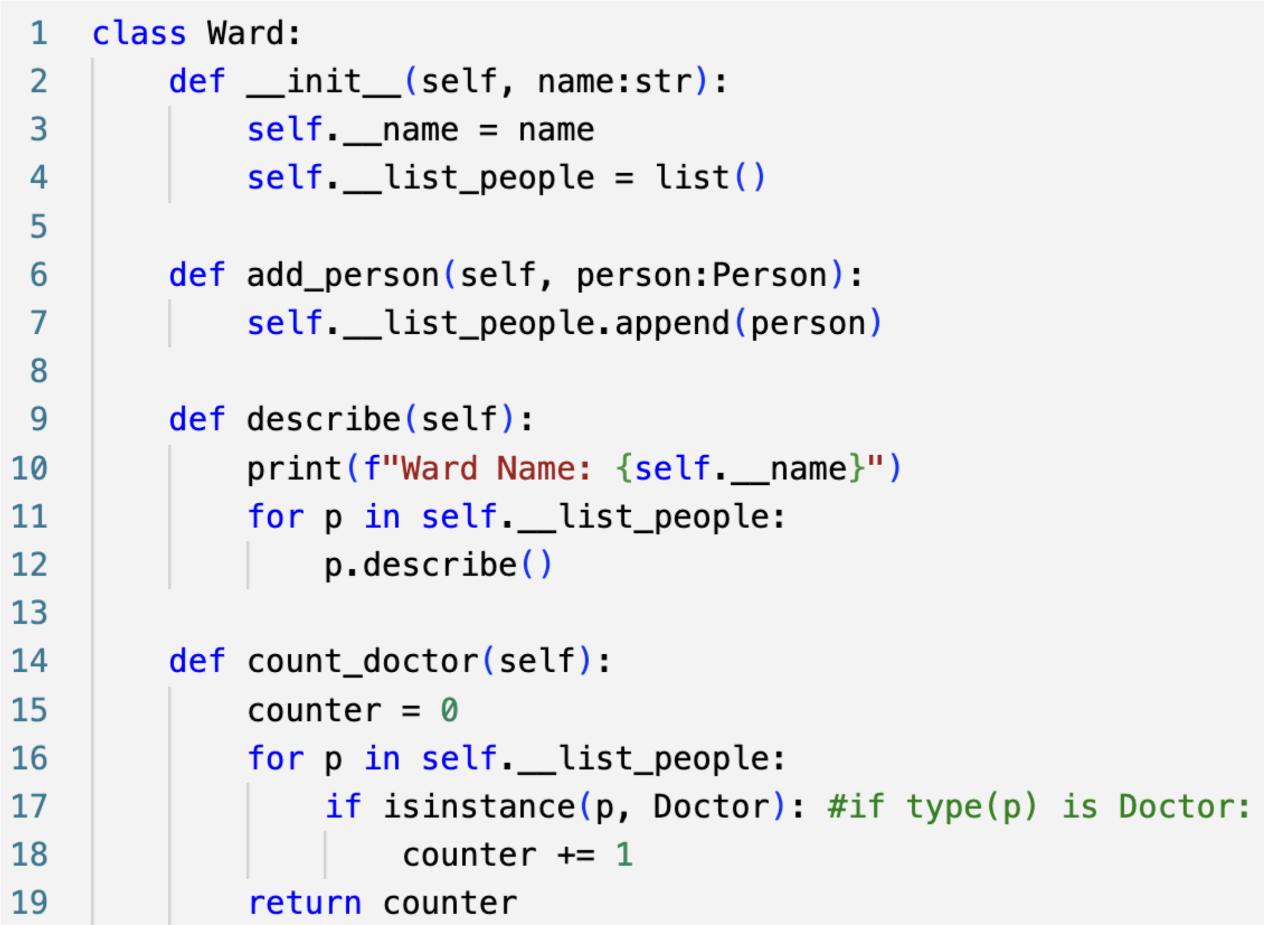
- name: string
- list_people(): list

+ add_person(): void
+ describe(): void
+ count_doctor(): int

Characteristics of OOP

Description

(c): count_doctor().

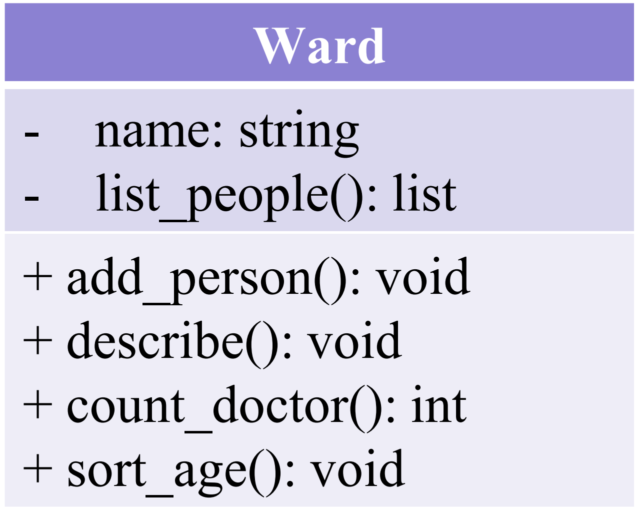


```python
class Ward:
    def __init__(self, name:str):
        self.__name = name
        self.__list_people = list()

    def add_person(self, person:Person):
        self.__list_people.append(person)

    def describe(self):
        print(f"Ward Name: {self.__name}")
        for p in self.__list_people:
            p.describe()

    def count_doctor(self):
        counter = 0
        for p in self.__list_people:
            if isinstance(p, Doctor): #if type(p) is Doctor:
                counter += 1
        return counter
```

Characteristics of OOP

Description

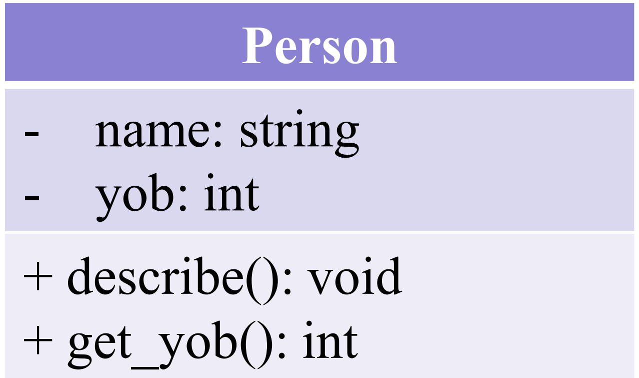
(d): sort_age(): Sorted by age (ASC)



Ward

- name: string
- list_people(): list

+ add_person(): void
+ describe(): void
+ count_doctor(): int
+ sort_age(): void



Person

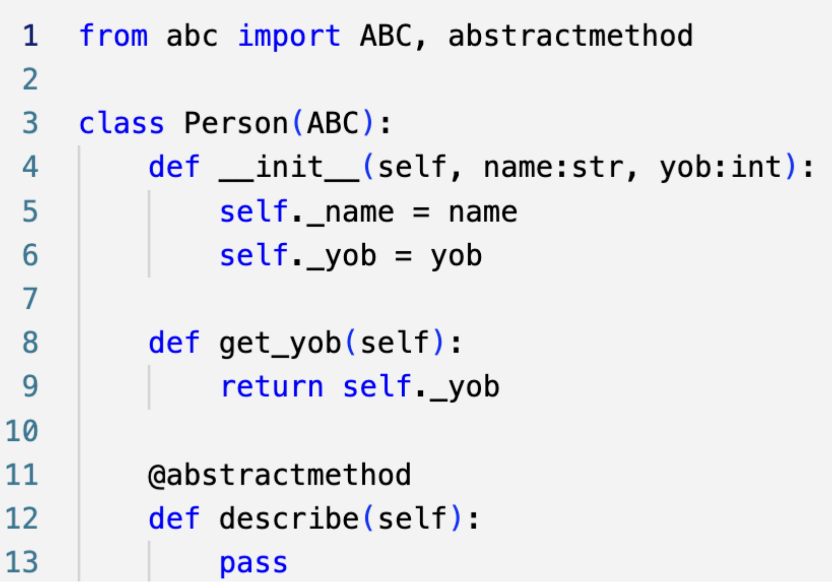
- name: string
- yob: int

+ describe(): void
+ get_yob(): int

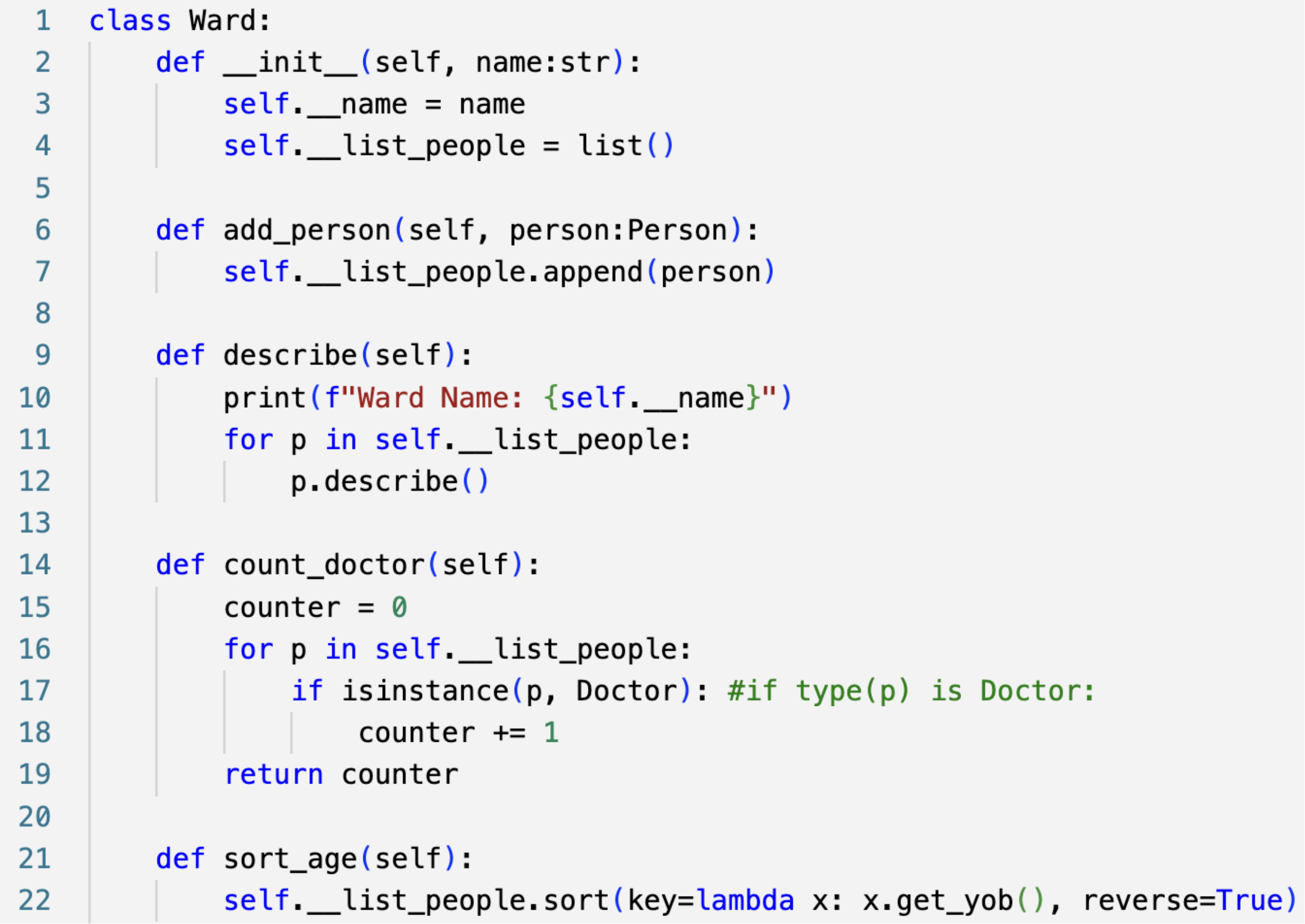
Characteristics of OOP

Description

(d): sort_age(): Sorted by age (ASC)



```python
from abc import ABC, abstractmethod

class Person(ABC):
    def __init__(self, name:str, yob:int):
        self._name = name
        self._yob = yob

    def get_yob(self):
        return self._yob

    @abstractmethod
    def describe(self):
        pass
```

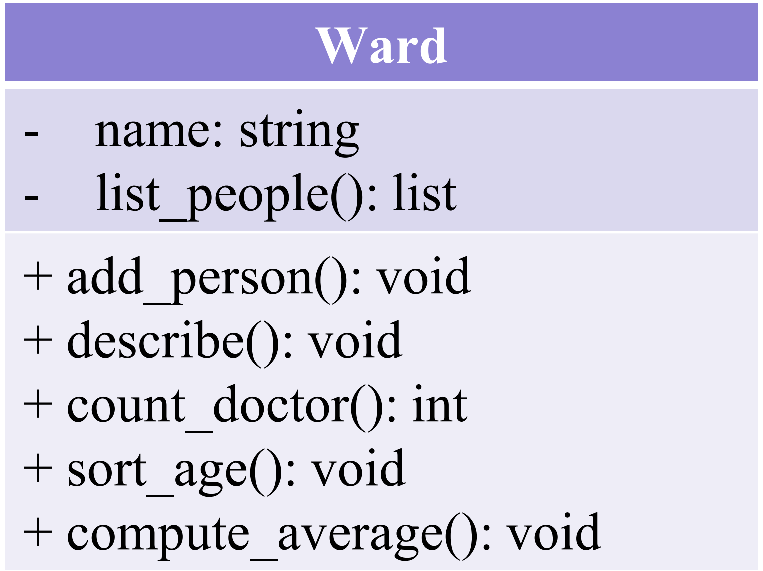


```python
class Ward:
    def __init__(self, name:str):
        self.__name = name
        self.__list_people = list()

    def add_person(self, person:Person):
        self.__list_people.append(person)

    def describe(self):
        print(f"Ward Name: {self.__name}")
        for p in self.__list_people:
            p.describe()

    def count_doctor(self):
        counter = 0
        for p in self.__list_people:
            if isinstance(p, Doctor): #if type(p) is Doctor:
                counter += 1
        return counter

    def sort_age(self):
        self.__list_people.sort(key=lambda x: x.get_yob(), reverse=True)
```

Characteristics of OOP

Description

(e): compute_average() method.



Ward

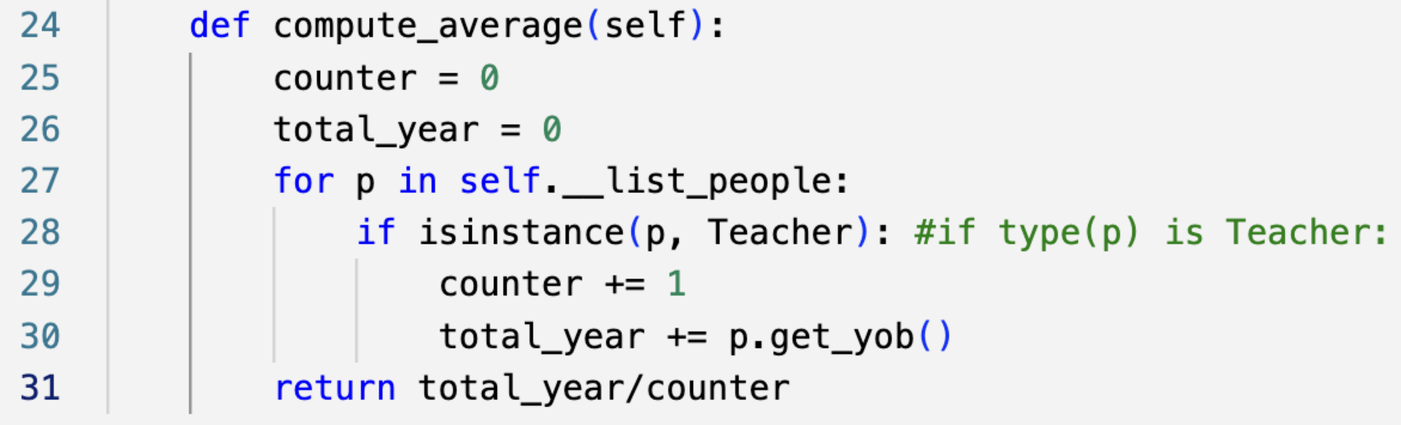
- name: string
- list_people(): list

+ add_person(): void
+ describe(): void
+ count_doctor(): int
+ sort_age(): void
+ compute_average(): void

Characteristics of OOP

Description

(e): compute_average() method.



```python
    def compute_average(self):
        counter = 0
        total_year = 0
        for p in self.__list_people:
            if isinstance(p, Teacher): #if type(p) is Teacher:
                counter += 1
                total_year += p.get_yob()
        return total_year/counter
```

Outline

SECTION 1
OOP Review

SECTION 2
OOP in Pytorch

SECTION 3
Characteristics of OOP

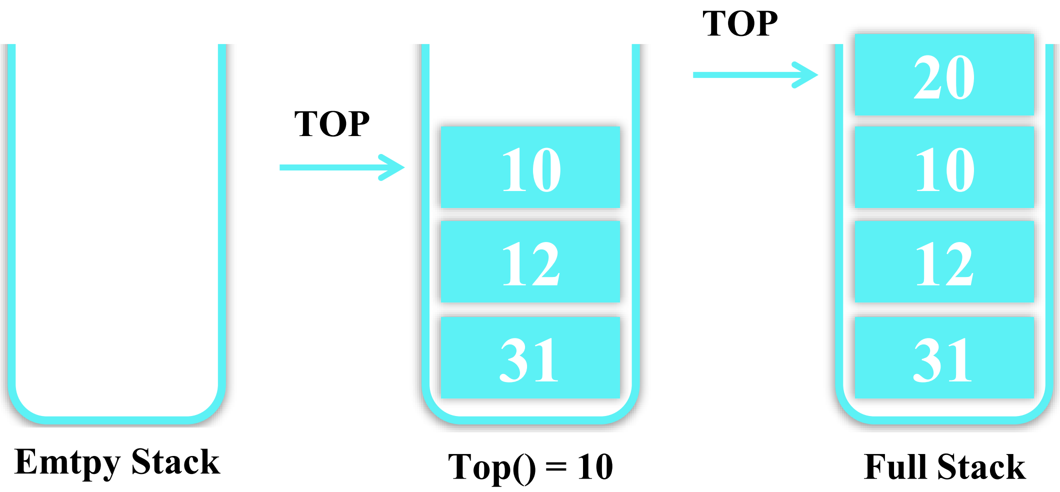
SECTION 4
Stack

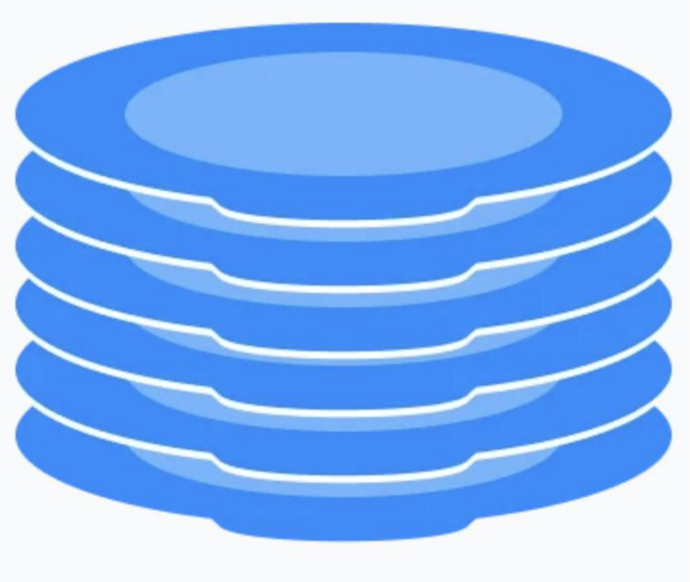
SECTION 4
Queue

Stack

Stack

- Last In First Out (LIFO)
- Pre-defined capacity (Limited size)

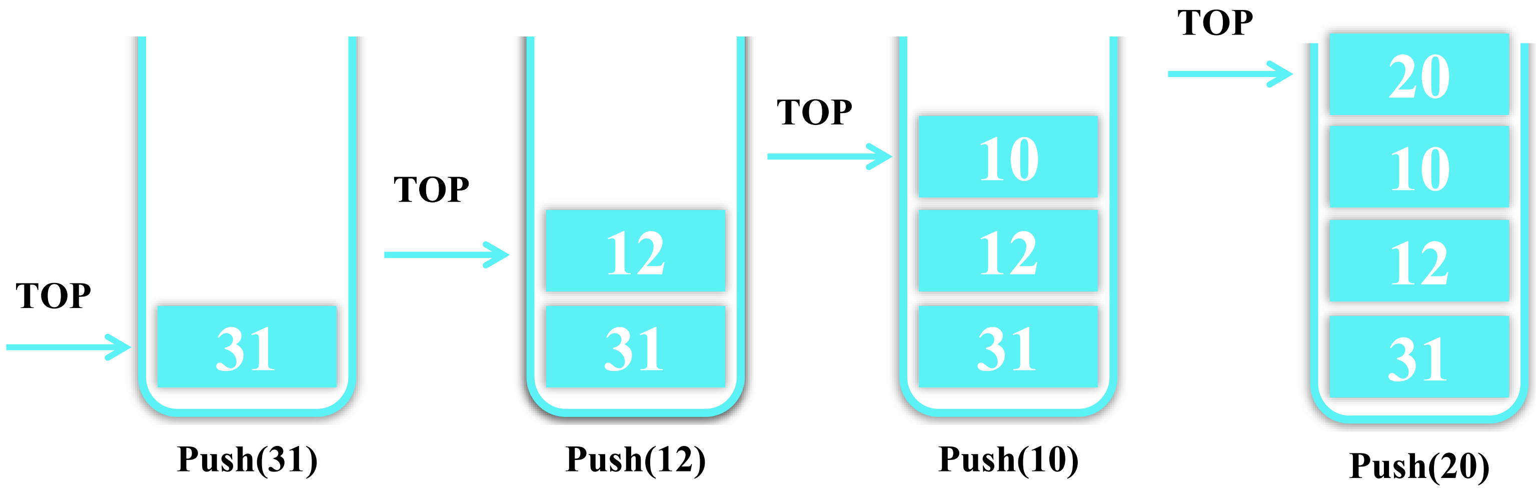




Stack

Operations

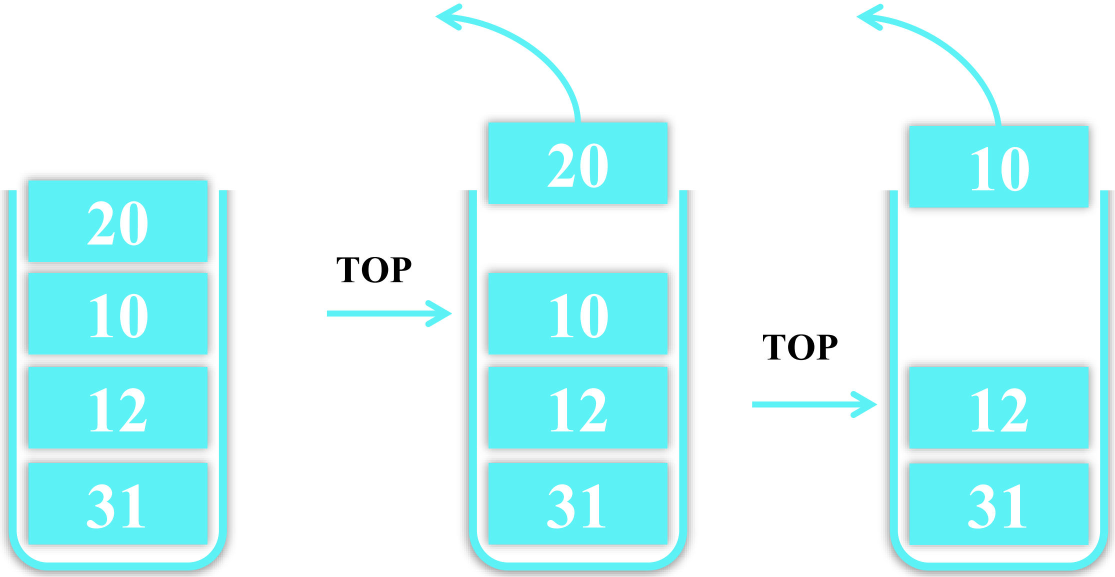
- Push: Add an element to the top of a stack



Stack

Operations

- Pop: Remove an element from the top of a stack

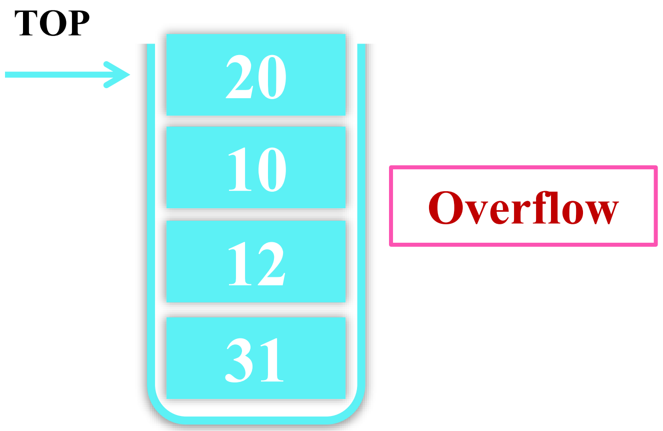


Push(20) Pop() Pop()

Stack

Operations

- Overflow: try to push an element to a full stack

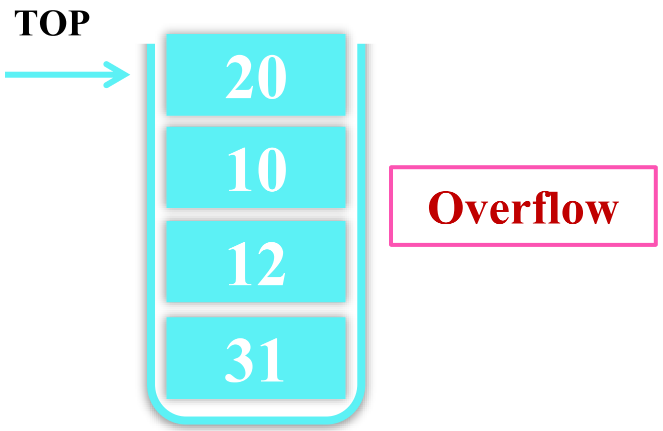


Push(20)

Stack

Operations

- Overflow: try to push an element to a full stack
- is_full: Check if the stack is full

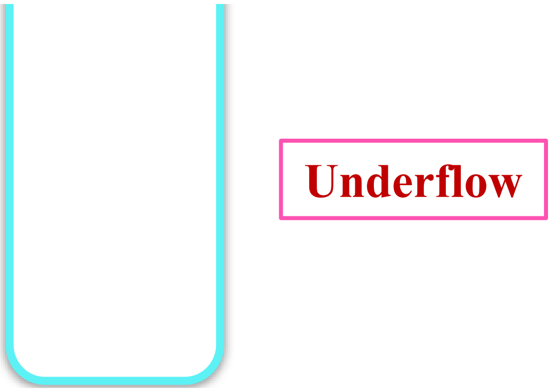


Push(20)

Stack

Operations

- Underflow: try to pop out an element to an empty stack

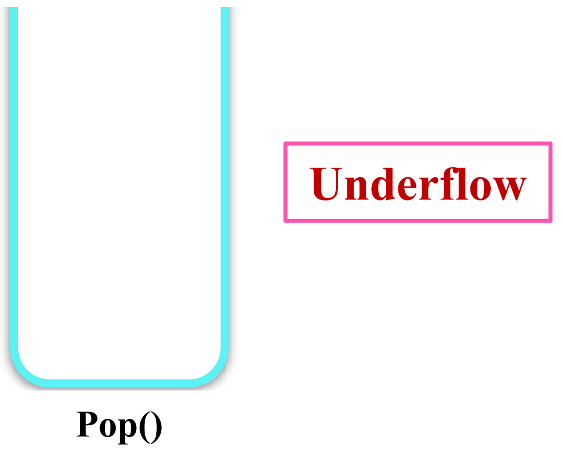


Pop()

Stack

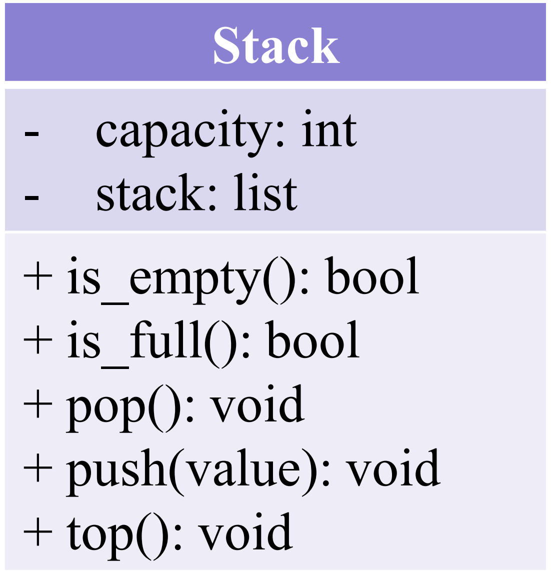
Operations

- Underflow: try to pop out an element to an empty stack
- is_empty: Check if the stack is empty



Stack

Description



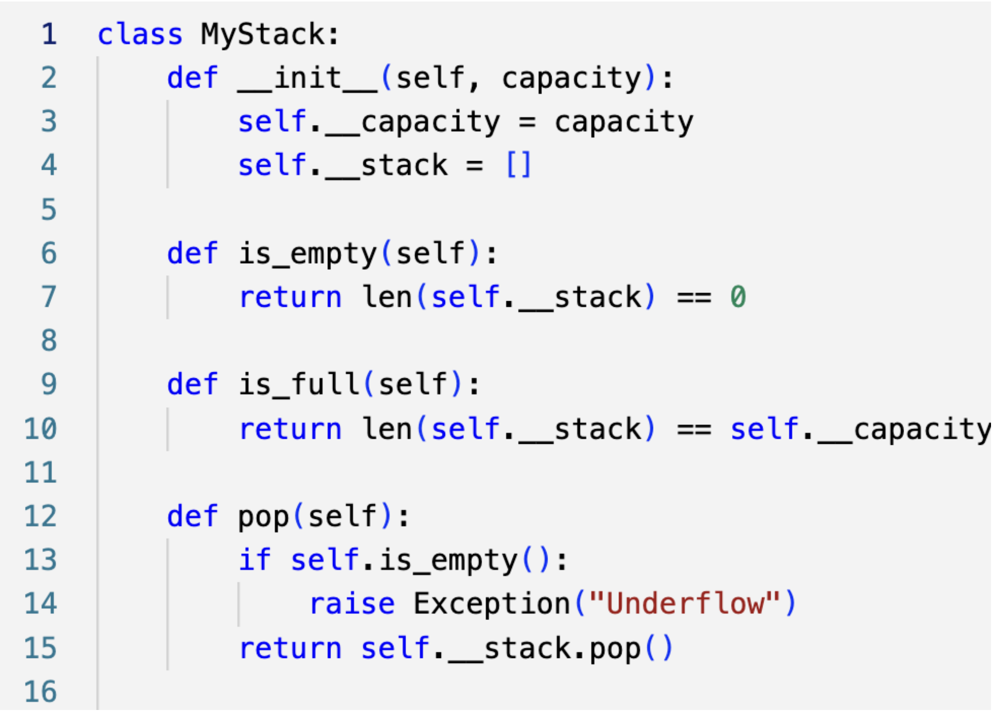
Stack

- capacity: int
- stack: list

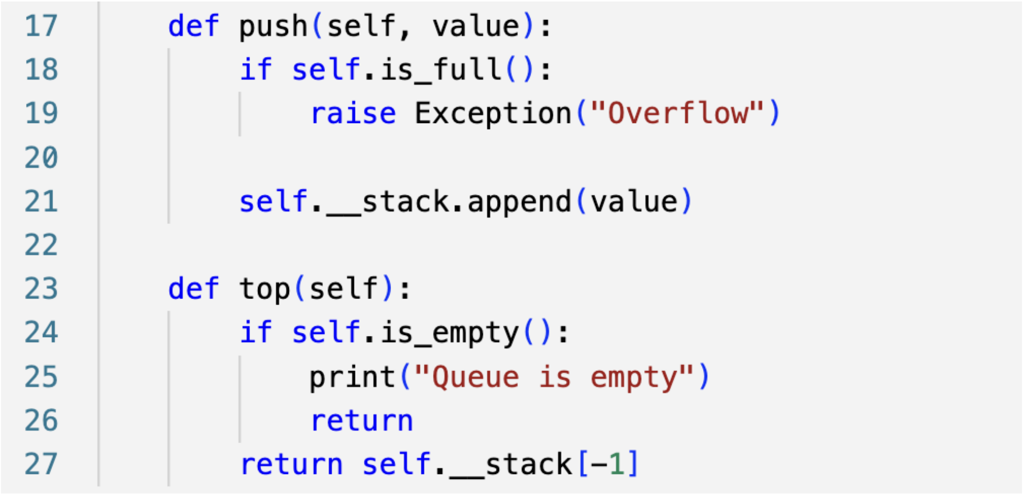
+ is_empty(): bool
+ is_full(): bool
+ pop(): void
+ push(value): void
+ top(): void

Stack

Solution



```python
class MyStack:
    def __init__(self, capacity):
        self.__capacity = capacity
        self.__stack = []

    def is_empty(self):
        return len(self.__stack) == 0

    def is_full(self):
        return len(self.__stack) == self.__capacity

    def pop(self):
        if self.is_empty():
            raise Exception("Underflow")
        return self.__stack.pop()

```



```python
    def push(self, value):
        if self.is_full():
            raise Exception("Overflow")

        self.__stack.append(value)

    def top(self):
        if self.is_empty():
            print("Queue is empty")
            return
        return self.__stack[-1]
```

Outline

SECTION 1
OOP Review

SECTION 2
OOP in Pytorch

SECTION 3
Characteristics of OOP

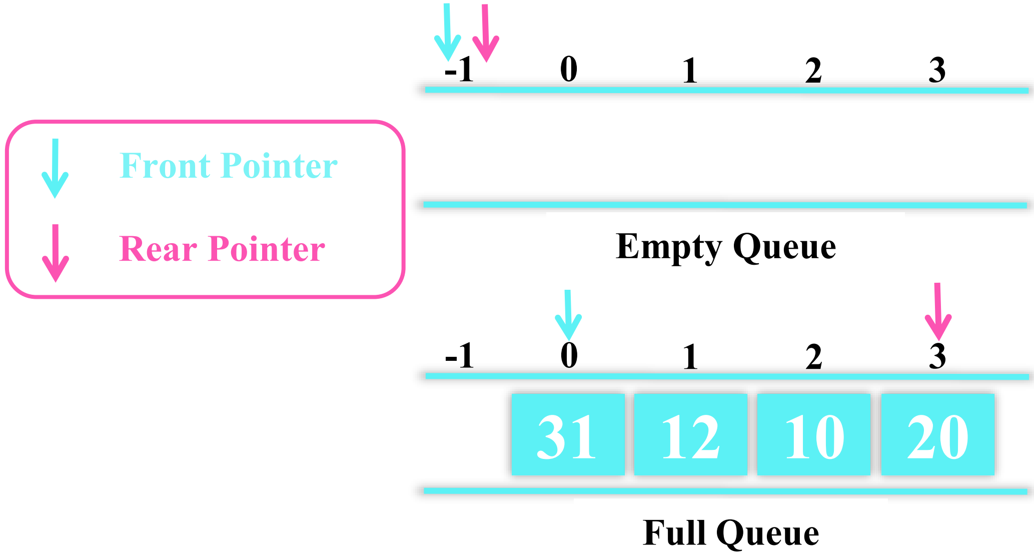
SECTION 4
Stack

SECTION 4
Queue

Queue

Queue

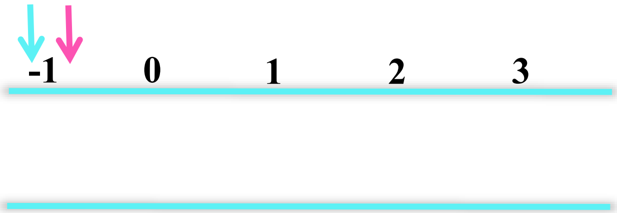
- First In First Out (FIFO)
- Pre-defined capacity (Limited size)



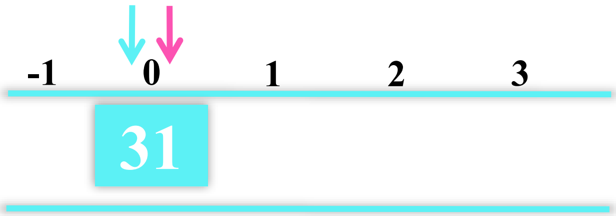
Queue

Operations

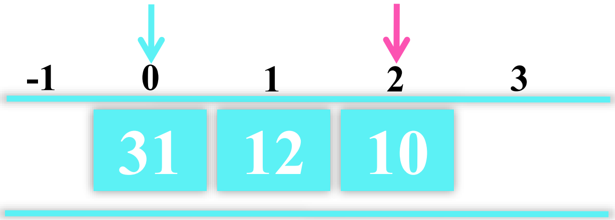
- Enqueue: Add an element to the end of the queue



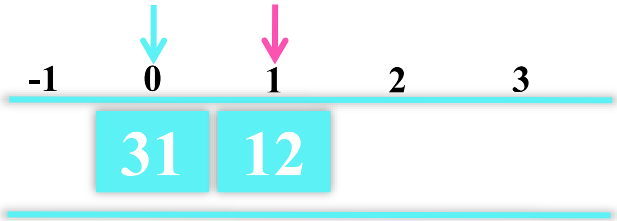
Empty Queue



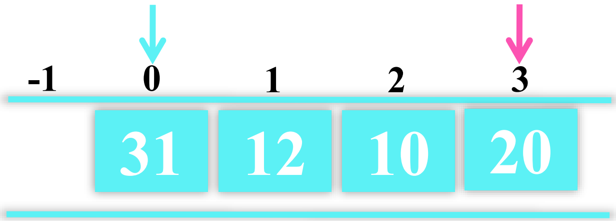
Enqueue(31)



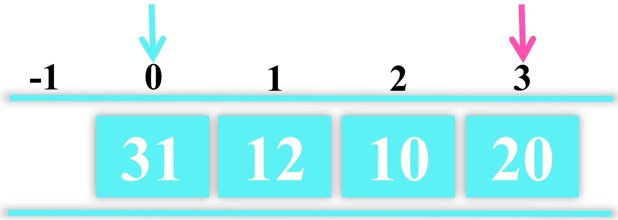
Enqueue(10)



Enqueue(12)



Enqueue(20)

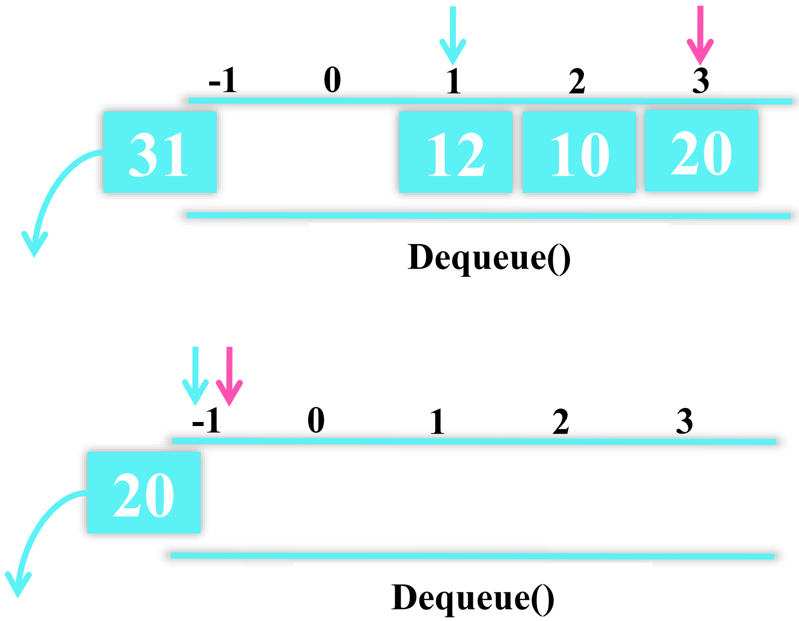


Full Queue

Queue

Operations

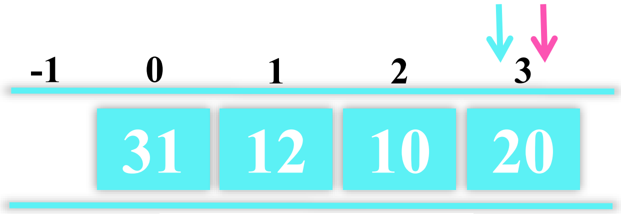
- Dequeue: Remove an element from the front of the queue



Queue

Operations

- Overflow: Try to enqueue an element to a full queue



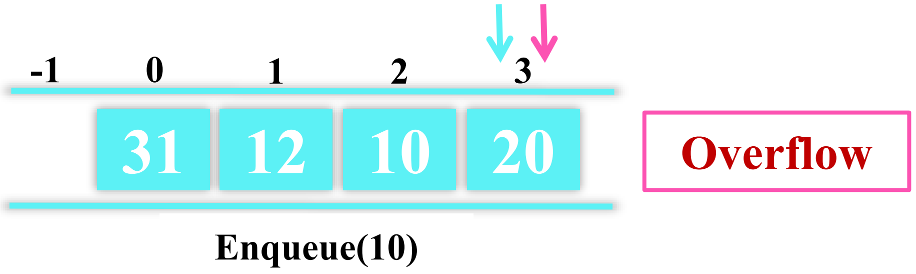
Overflow

Enqueue(10)

Queue

Operations

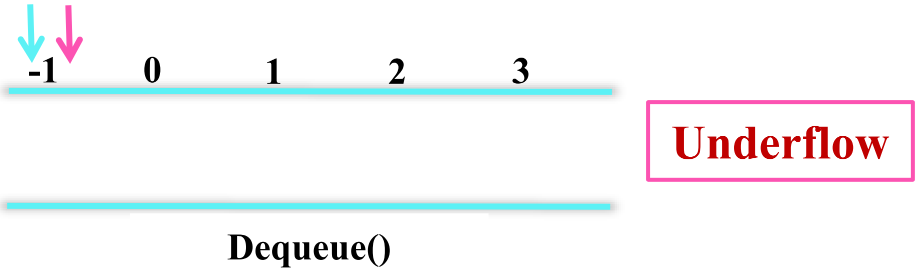
- Overflow: Try to enqueue an element to a full queue
- is_full: Check if the queue if full



Queue

Operations

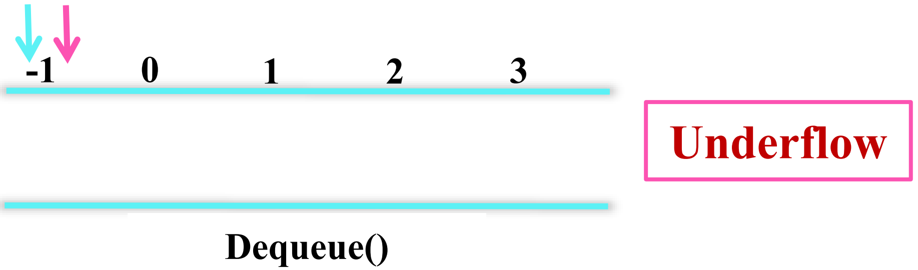
- Underflow: Try to dequeue an empty queue



Queue

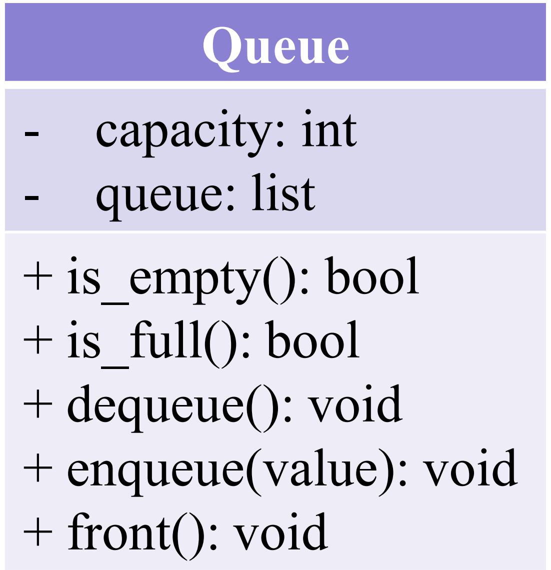
Operations

- Underflow: Try to dequeue an empty queue
- is_empty: Check if the queue is empty



Stack

Description



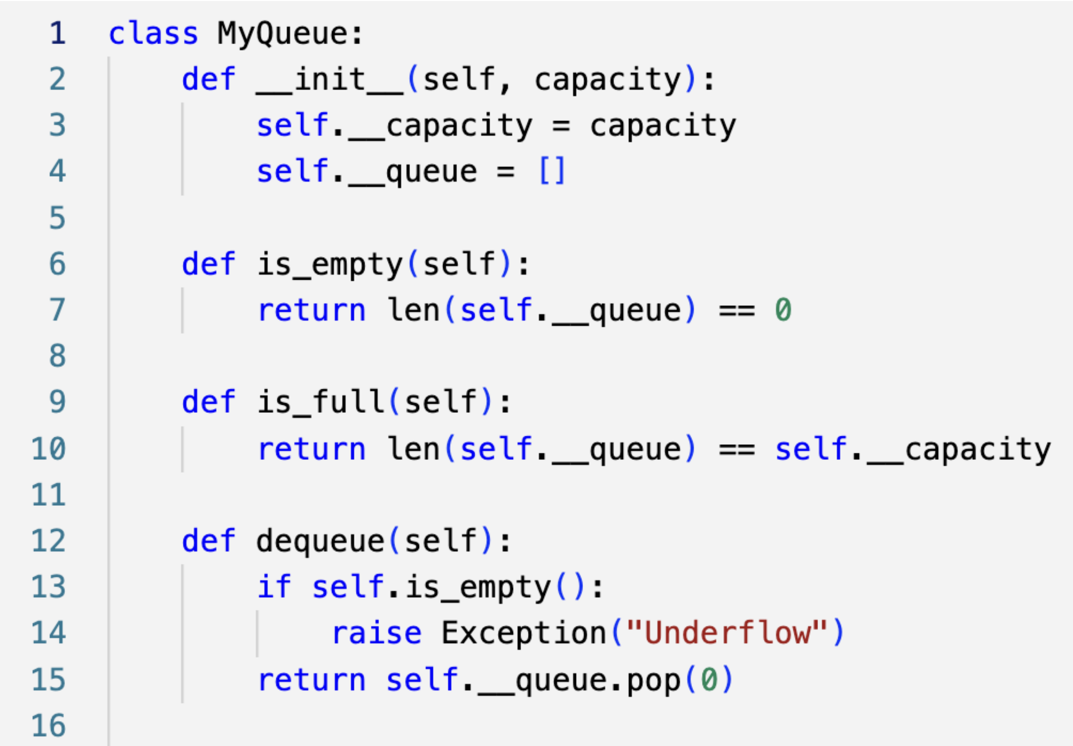
Queue

- capacity: int
- queue: list

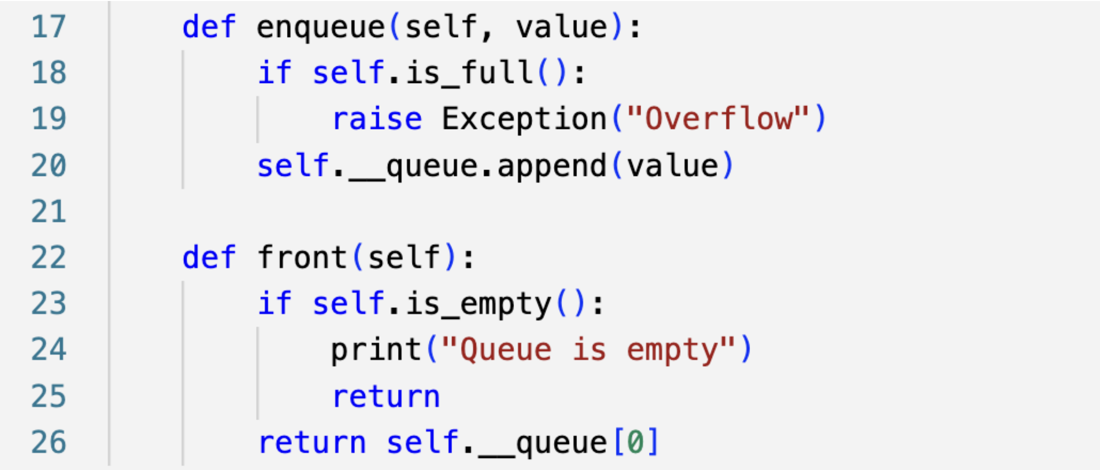
+ is_empty(): bool
+ is_full(): bool
+ dequeue(): void
+ enqueue(value): void
+ front(): void

Stack

Solution



```python
class MyQueue:
    def __init__(self, capacity):
        self.__capacity = capacity
        self.__queue = []

    def is_empty(self):
        return len(self.__queue) == 0

    def is_full(self):
        return len(self.__queue) == self.__capacity

    def dequeue(self):
        if self.is_empty():
            raise Exception("Underflow")
        return self.__queue.pop(0)

```



```python
    def enqueue(self, value):
        if self.is_full():
            raise Exception("Overflow")
        self.__queue.append(value)

    def front(self):
        if self.is_empty():
            print("Queue is empty")
            return
        return self.__queue[0]
```

Summary

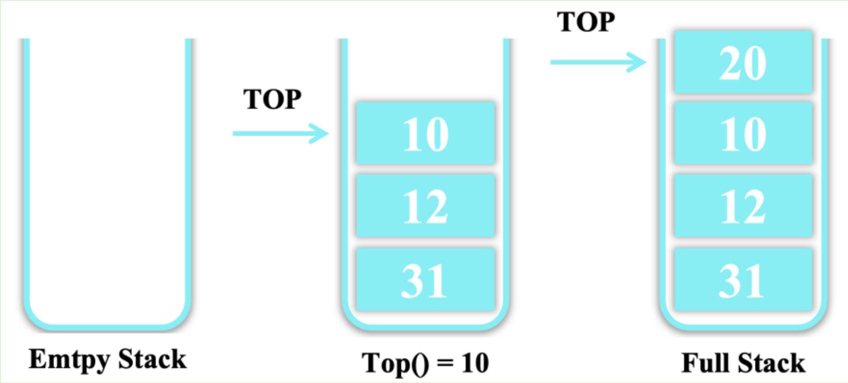
Softmax

$$\mathrm{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$$

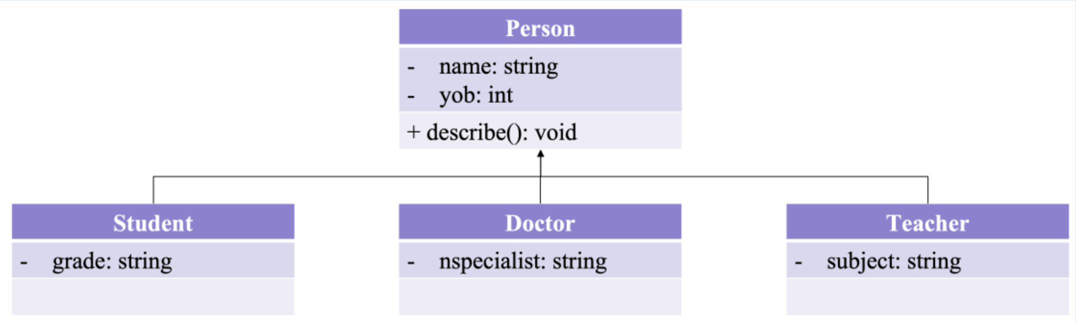
$$\mathrm{softmax\_stable}(x_i) = \frac{\exp(x_i - c)}{\sum_{j=1}^{n} \exp\left(x_j - c\right)}$$

$$c = \max(x)$$

Stack



OOP (User Managemnet)



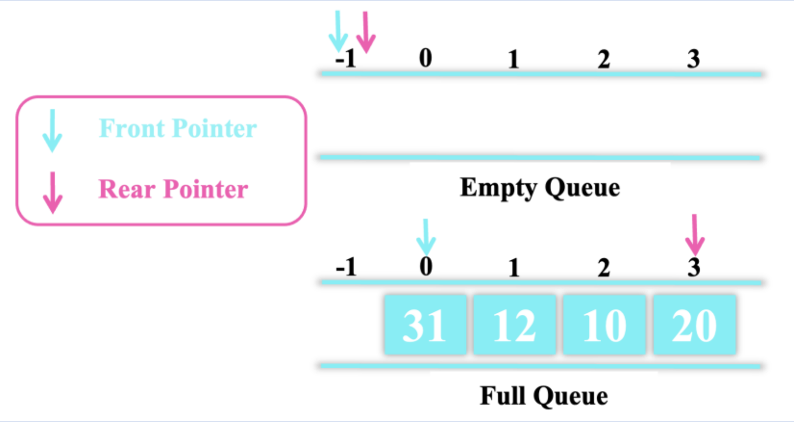
Person
- name: string
- yob: int
+ describe(): void

Student
- grade: string

Doctor
- nspecialist: string

Teacher
- subject: string

Queue



Thanks!

Any questions?