



Module 01 – Exercise Class

Object-Oriented Programming

Nguyen Quoc Thai

Objectives

OOP

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

Exercise

- ❖ nn.Module Pytorch
- ❖ Softmax
- ❖ 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 → `class Dog:`

Constructor → `def __init__(self, name, size, age, color):`

Attributes → `self.name = name`
`self.size = size`
`self.age = age`
`self.color = color`

Method → `def eat(self):`
`if self.age <= 1:`
`return 'Chicken'`
`else:`
`return 'Fish'`



Encapsulation

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

```
1 class Dog:
2     def __init__(self, name, size, age):
3         self.name = name
4         self._size = size
5         self.__age = age
6
7 dog_1 = Dog('Chow Chow', 'Small', 2)
8 print(dog_1.name)
9 print(dog_1._size)
10 print(dog_1.__age)
```

Chow Chow
Small

```
AttributeError                                Traceback (most recent call last)
<ipython-input-36-561eace23900> in <cell line: 10>()
      8 print(dog_1.name)
      9 print(dog_1._size)
----> 10 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

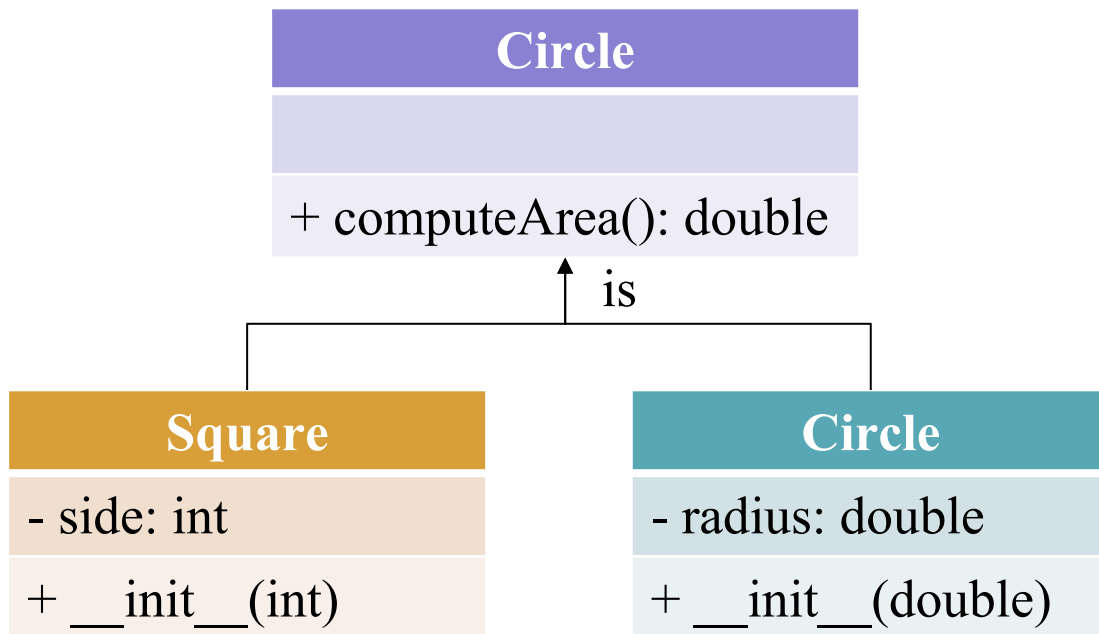
```
1 class Dog:
2     def __init__(self, name):
3         self.__name = name
4
5     def getName(self):
6         return self.__name
7
8     def setName(self, name):
9         self.__name = name
10
11 dog_1 = Dog('Chow Chow')
12 print(dog_1.getName())
13 dog_1.setName('Chaw Chaw')
14 print(dog_1.getName())
```

Chow Chow
Chaw Chaw



Abstraction

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



```
1 from abc import ABC, abstractmethod
2
3 class Shape(ABC):
4     @abstractmethod
5     def computeArea(self):
6         pass
7
8 class Square(Shape):
9     def __init__(self, side):
10         self.__side = side
11
12     def computeArea(self):
13         return self.__side*self.__side
14
15 square = Square(5)
16 print(square.computeArea())
```



Inheritance

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

Base Class

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

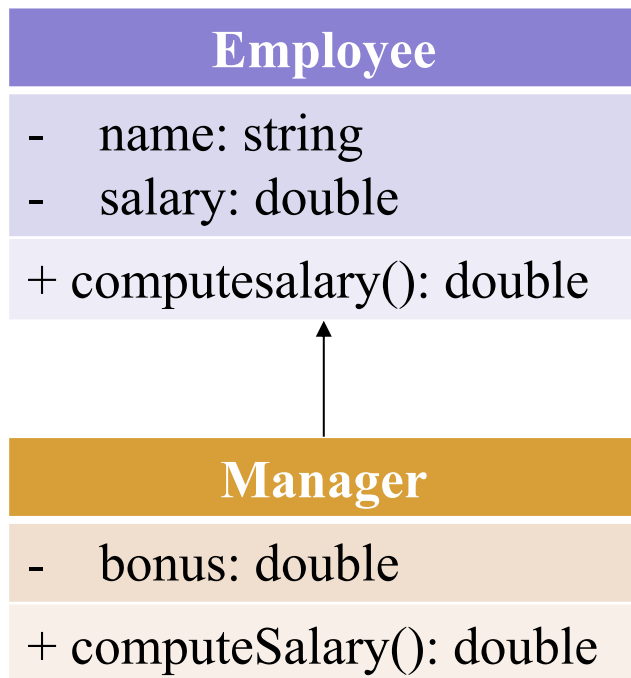
Derived Class

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

Review



Inheritance

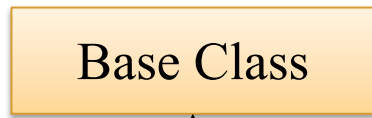


```
1 class Employee:
2     def __init__(self, name, salary):
3         self._name = name
4         self._salary = salary
5
6     def computeSalary(self):
7         return self._salary
8
9 class Manager(Employee):
10    def __init__(self, name, salary, bonus):
11        self._name = name
12        self._salary = salary
13        self.__bonus = bonus
14
15    def computeSalary(self):
16        return super().computeSalary() + self.__bonus
```

```
1 mai = Manager('Mai', 100, 50)
2 salary = mai.computeSalary()
3 print(salary)
```



Inheritance



Derived Class

Single
Inheritance

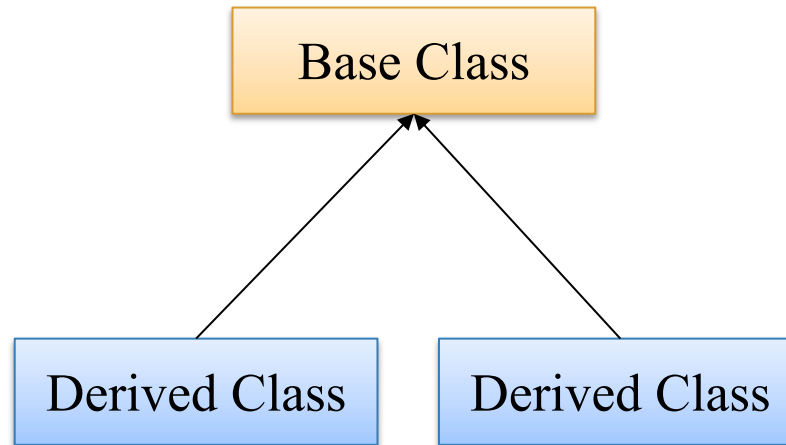


Derived Class



Derived Class

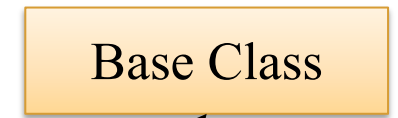
Multilevel
Inheritance



Derived Class

Derived Class

Hierarchical
Inheritance



Derived Class

Multiple
Inheritance



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)

```
1 class A:
2     def __init__(self, num):
3         self.num = num
4
5     def show(self):
6         print(self.num)
7
8 class B(A):
9     def show(self):
10        print(self.num*self.num)
11
12 ins_B = B(3)
13 ins_B.show()
```



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 softmax và stable softmax theo công thức sau:

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

$$\begin{aligned} \text{softmax_stable}(x_i) &= \frac{\exp(x_i - c)}{\sum_{j=1}^n \exp(x_j - c)} \\ c &= \max(x) \end{aligned}$$

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



Softmax



OOP in Pytorch



Softmax

1	0.09
2	0.24
3	0.67

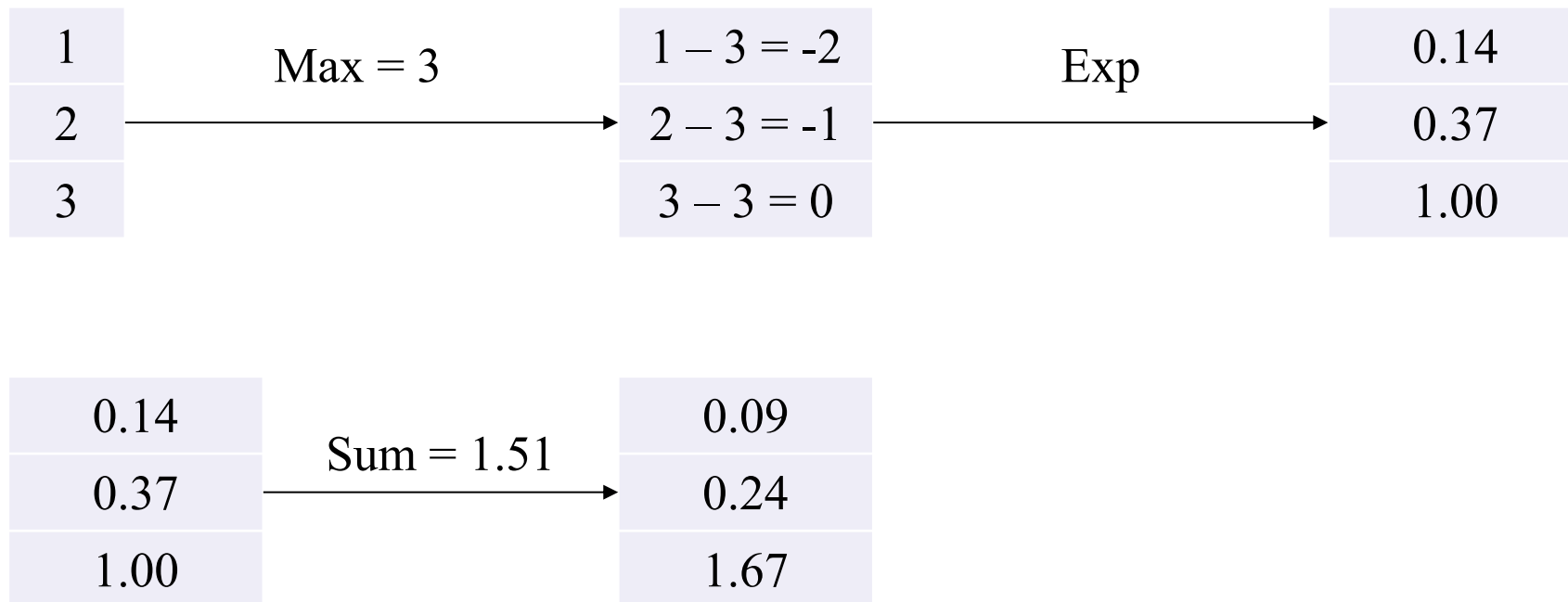
```
1 import torch
2 import torch.nn as nn
3
4 class MySoftmax(nn.Module):
5     def __init__(self):
6         super().__init__()
7
8     def forward(self, x):
9         x_exp = torch.exp(x)
10        partition = x_exp.sum(0, keepdims=True)
11        return x_exp / partition
12
13 data = torch.Tensor([1, 2, 3])
14 my_softmax = MySoftmax()
15 output = my_softmax(data)
16 output
```

tensor([0.0900, 0.2447, 0.6652])

OOP in Pytorch



Stable Softmax



OOP in Pytorch



Stable Softmax

1	0.09
2	0.24
3	1.67

```
1 import torch
2 import torch.nn as nn
3
4 class SoftmaxStable(nn.Module):
5     def __init__(self):
6         super().__init__()
7
8     def forward(self, x):
9         x_max = torch.max(x, dim=0, keepdims=True)
10        x_exp = torch.exp(x - x_max.values)
11        partition = x_exp.sum(0, keepdims=True)
12        return x_exp / partition
13
14 data = torch.Tensor([1, 2, 3])
15 softmax_stable = SoftmaxStable()
16 output = softmax_stable(data)
17 output
```

tensor([0.0900, 0.2447, 0.6652])



Outline

SECTION 1

OOP Review

SECTION 4

Stack

SECTION 2

OOP in Pytorch

SECTION 4

Queue

SECTION 3

Characteristics of OOP

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
- listPeople(): list

Characteristics of OOP



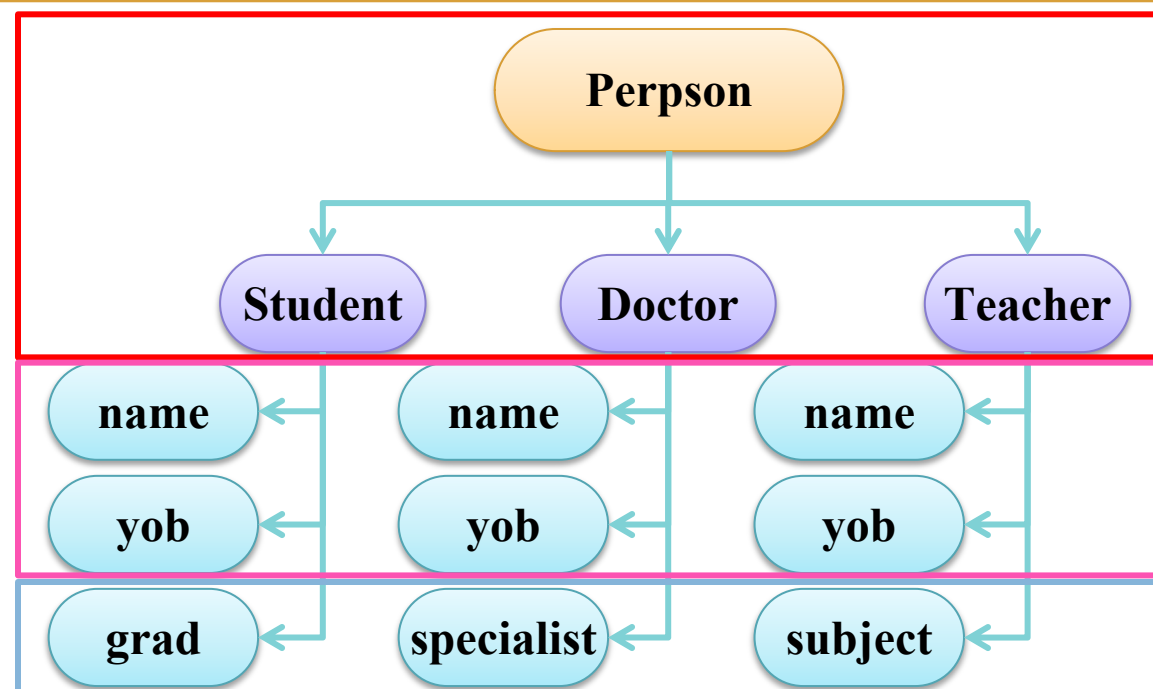
Description

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.

Is-a relationship

Same attributes

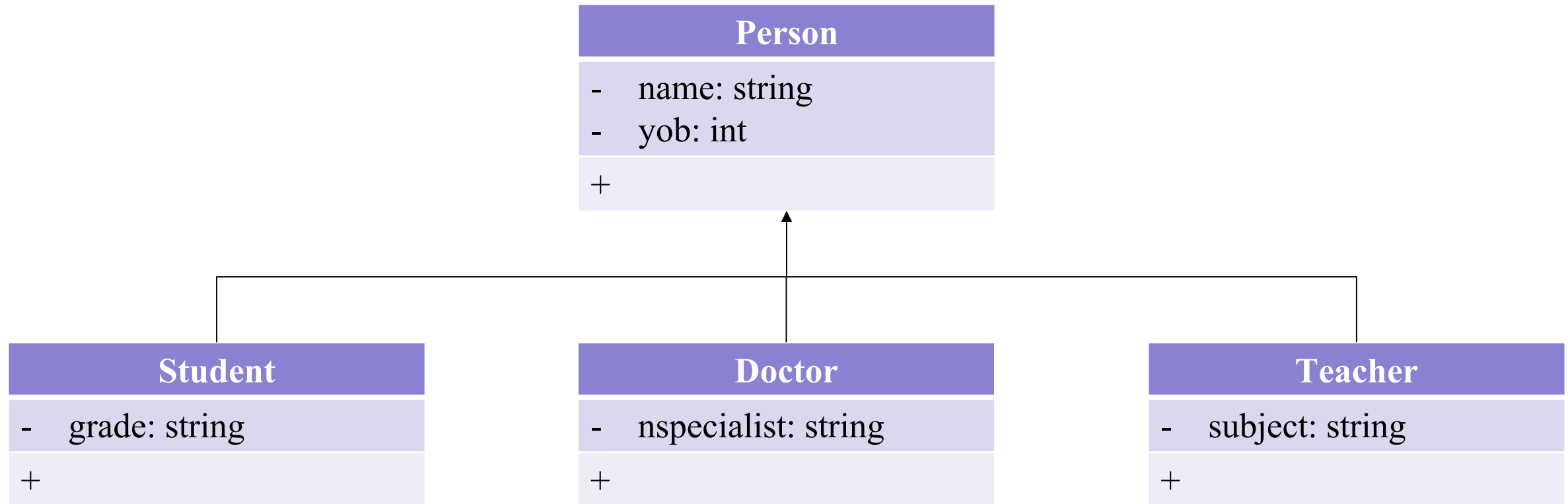
Unique attributes



Characteristics of OOP



Description

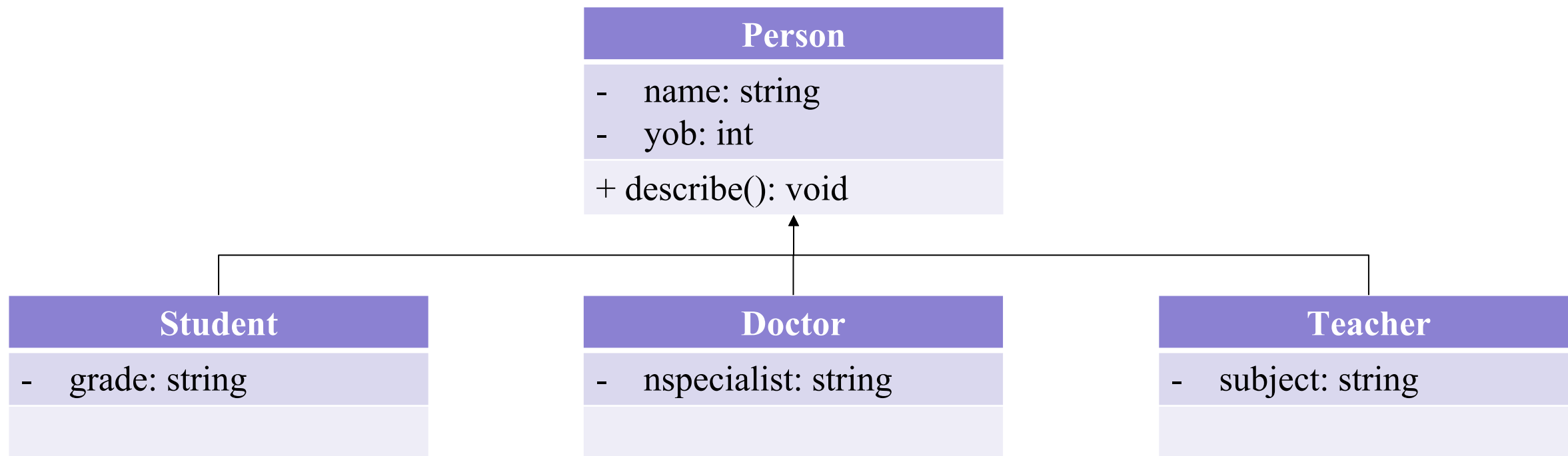


Characteristics of OOP



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.

```
1 from abc import ABC, abstractmethod
2
3 class Person(ABC):
4     def __init__(self, name:str, yob:int):
5         self._name = name
6         self._yob = yob
7
8     def getYoB(self):
9         return self._yob
10
11     @abstractmethod
12     def describe(self):
13         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.

```
16 class Student(Person):
17     def __init__(self, name:str, yob:int, grade:str):
18         super().__init__(name=name, yob=yob)
19         self.__grade = grade
20
21     def describe(self):
22         print(f"Student - Name: {self._name} - YoB: {self._yob} - Grade: {self.__grade}")
23
24
25 class Teacher(Person):
26     def __init__(self, name:str, yob:int, subject:str):
27         super().__init__(name=name, yob=yob)
28         self.__subject = subject
29
30     def describe(self):
31         print(f"Teacher - Name: {self._name} - YoB: {self._yob} - Subject: {self.__subject}")
32
33
34 class Doctor(Person):
35     def __init__(self, name:str, yob:int, specialist:str):
36         super().__init__(name=name, yob=yob)
37         self.__specialist = specialist
38
39     def describe(self):
40         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.

```
1 student1 = Student(name="studentA", yob=2010, grade="7")
2 student1.describe()
3
4 teacher1 = Teacher(name="teacherA", yob=1969, subject="Math")
5 teacher1.describe()
6
7 doctor1 = Doctor(name="doctorA", yob=1945, specialist="Endocrinologists")
8 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): addPerson(person) method.

Ward
- name: string
- listPeople(): list
+ addPerson(): void
+ describe(): void

Characteristics of OOP



Description

(b): addPerson(person) method.

```
1 class Ward:
2     def __init__(self, name:str):
3         self.__name = name
4         self.__listPeople = list()
5
6     def addPerson(self, person:Person):
7         self.__listPeople.append(person)
8
9     def describe(self):
10        print(f"Ward Name: {self.__name}")
11        for p in self.__listPeople:
12            p.describe()
```

Characteristics of OOP



Description

(b): addPerson(person) method.

```
1 student1 = Student(name="studentA", yob=2010, grade="7")
2 teacher1 = Teacher(name="teacherA", yob=1969, subject="Math")
3 doctor1 = Doctor(name="doctorA", yob=1945, specialist="Endocrinologists")
4 teacher2 = Teacher(name="teacherB", yob=1995, subject="History")
5 doctor2 = Doctor(name="doctorB", yob=1975, specialist="Cardiologists")
6 ward1 = Ward(name="Ward1")
7 ward1.addPerson(student1)
8 ward1.addPerson(teacher1)
9 ward1.addPerson(teacher2)
10 ward1.addPerson(doctor1)
11 ward1.addPerson(doctor2)
12 ward1.describe()
```

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): countDoctor().

Ward
- name: string
- listPeople(): list
+ addPerson(): void
+ describe(): void
+ countDoctor(): int

Characteristics of OOP



Description

(c): countDoctor().

```
1 class Ward:
2     def __init__(self, name:str):
3         self.__name = name
4         self.__listPeople = list()
5
6     def addPerson(self, person:Person):
7         self.__listPeople.append(person)
8
9     def describe(self):
10        print(f"Ward Name: {self.__name}")
11        for p in self.__listPeople:
12            p.describe()
13
14    def countDoctor(self):
15        counter = 0
16        for p in self.__listPeople:
17            if isinstance(p, Doctor): #if type(p) is Doctor:
18                counter += 1
19        return counter
```

Characteristics of OOP



Description

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

Ward

- name: string
- listPeople(): list
- + addPerson(): void
- + describe(): void
- + countDoctor(): int
- + sortAge(): void

Person

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

Characteristics of OOP



Description

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

```
1 from abc import ABC, abstractmethod
2
3 class Person(ABC):
4     def __init__(self, name:str, yob:int):
5         self._name = name
6         self._yob = yob
7
8     def getYoB(self):
9         return self._yob
10
11     @abstractmethod
12     def describe(self):
13         pass
```

```
1 class Ward:
2     def __init__(self, name:str):
3         self.__name = name
4         self.__listPeople = list()
5
6     def addPerson(self, person:Person):
7         self.__listPeople.append(person)
8
9     def describe(self):
10        print(f"Ward Name: {self.__name}")
11        for p in self.__listPeople:
12            p.describe()
13
14    def countDoctor(self):
15        counter = 0
16        for p in self.__listPeople:
17            if isinstance(p, Doctor): #if type(p) is Doctor:
18                counter += 1
19        return counter
20
21    def sortAge(self):
22        self.__listPeople.sort(key=lambda x: x.getYoB(), reverse=True)
```

Characteristics of OOP



Description

(e): aveTeacherYearOfBirth() method.

Ward
- name: string
- listPeople(): list
+ addPerson(): void
+ describe(): void
+ countDoctor(): int
+ sortAge(): void
+ aveTeacherYearOfBirth(): void

Characteristics of OOP



Description

(e): aveTeacherYearOfBirth() method.

```
def aveTeacherYearOfBirth(self):  
    counter = 0  
    total_year = 0  
    for p in self.__listPeople:  
        if isinstance(p, Teacher): #if type(p) is Teacher:  
            counter += 1  
            total_year += p.getYoB()  
    return total_year/counter
```



Outline

SECTION 1

OOP Review

SECTION 2

OOP in Pytorch

SECTION 3

Characteristics of OOP

SECTION 4

Stack

SECTION 4

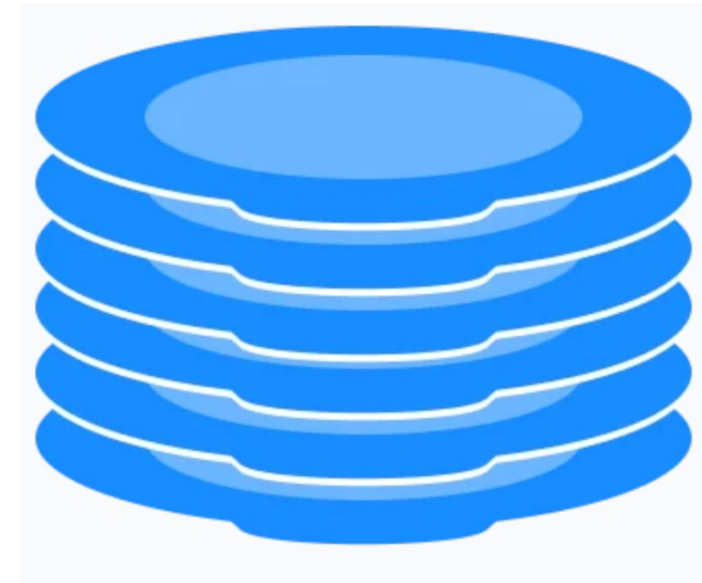
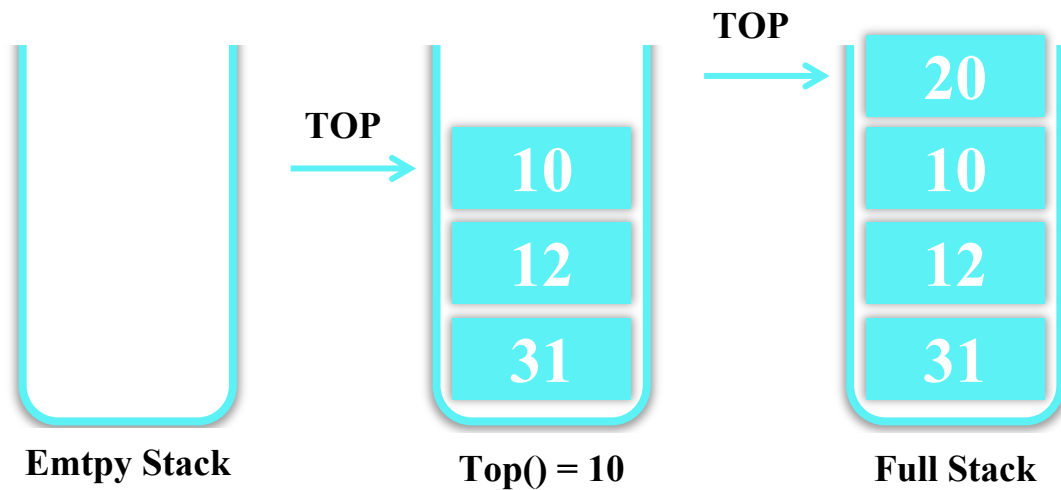
Queue

Stack



Stack

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

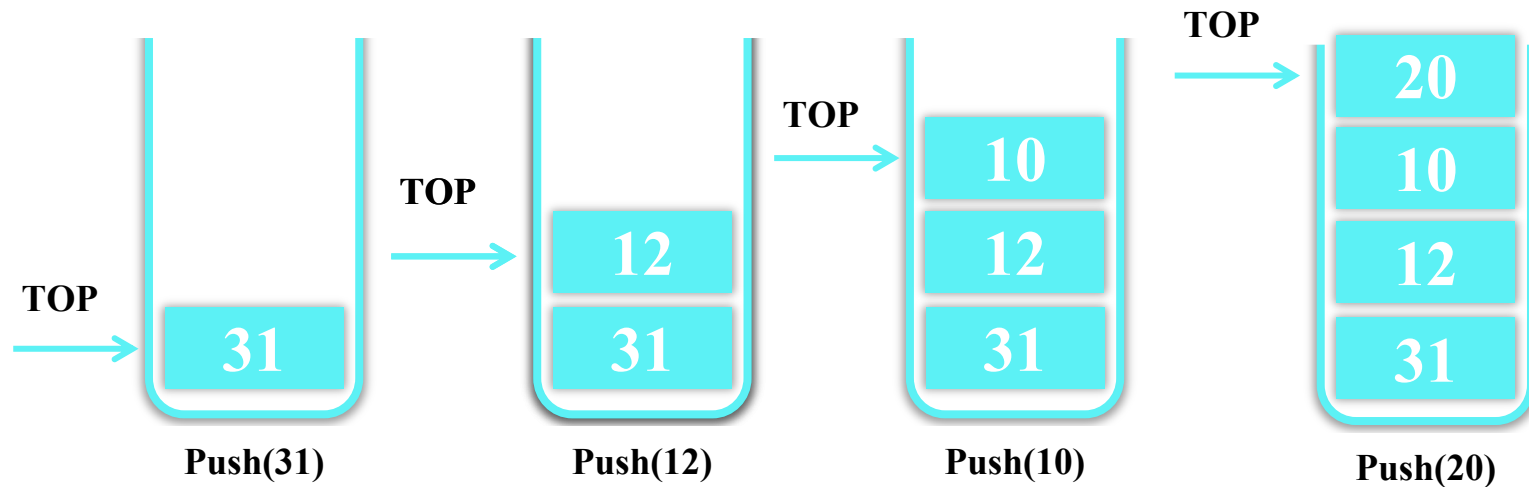


Stack



Operations

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

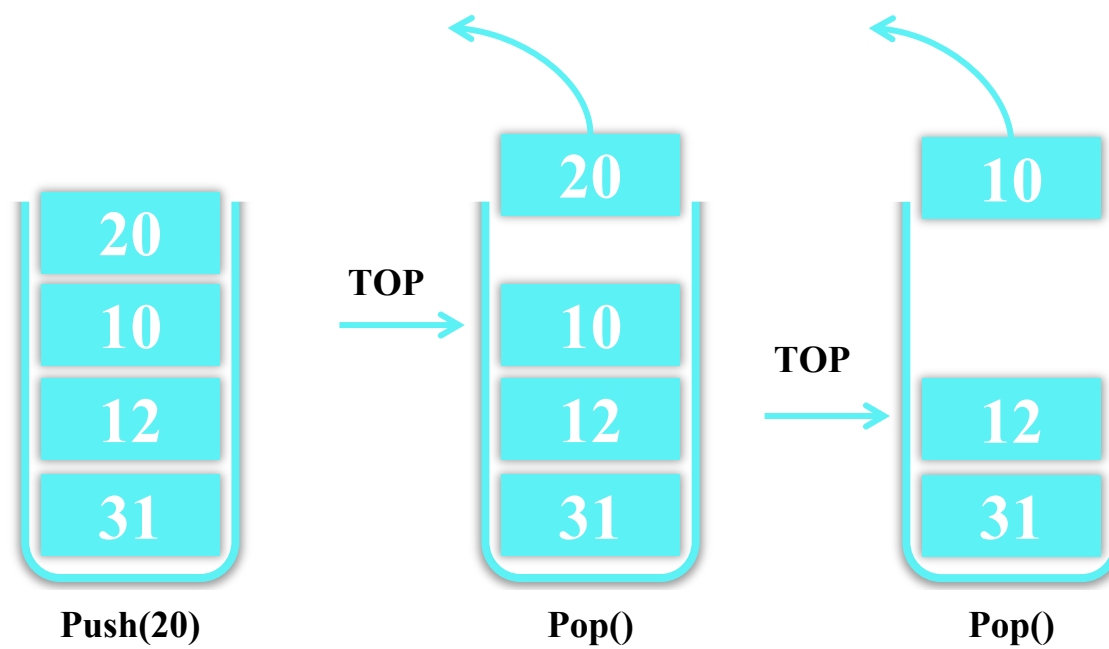


Stack



Operations

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

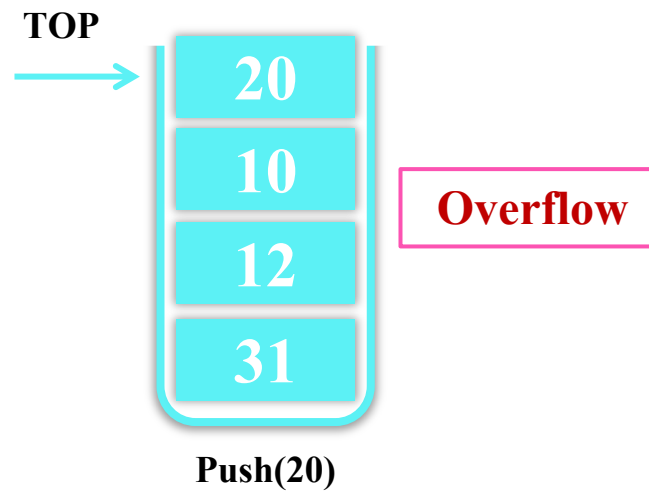


Stack



Operations

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

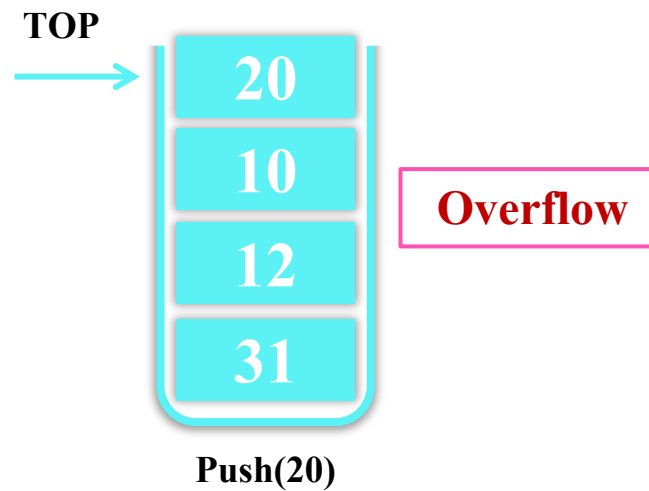


Stack



Operations

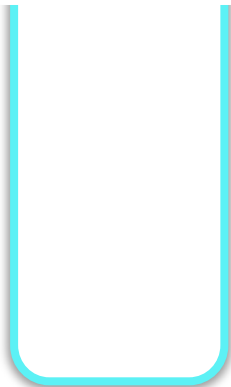
- Overflow: try to push an element to a full stack
- IsFull: Check if the stack is full





Operations

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



Pop()

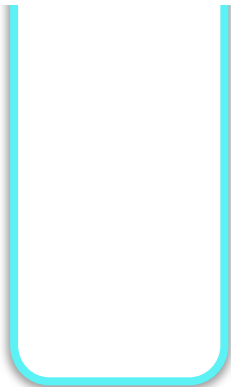
Underflow

Stack



Operations

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



Pop()

Underflow



Description

Stack
<ul style="list-style-type: none">- capacity: int- stack: list
<ul style="list-style-type: none">+ isEmpty(): bool+ isFull(): bool+ pop(): void+ push(value): void+ top(): void



Solution

```
1 class MyStack:
2     def __init__(self, capacity):
3         self.__capacity = capacity
4         self.__stack = []
5
6     def isEmpty(self):
7         return len(self.__stack) == 0
8
9     def isFull(self):
10        return len(self.__stack) == self.__capacity
11
12    def pop(self):
13        if self.isEmpty():
14            raise Exception("Underflow")
15        return self.__stack.pop()
```

```
16
17    def push(self, value):
18        if self.isFull():
19            raise Exception("Overflow")
20
21        self.__stack.append(value)
22
23    def top(self):
24        if self.isEmpty():
25            print("Queue is empty")
26            return
27        return self.__stack[-1]
```




Outline

SECTION 1

OOP Review

SECTION 2

OOP in Pytorch

SECTION 3

Characteristics of OOP

SECTION 4

Stack

SECTION 4

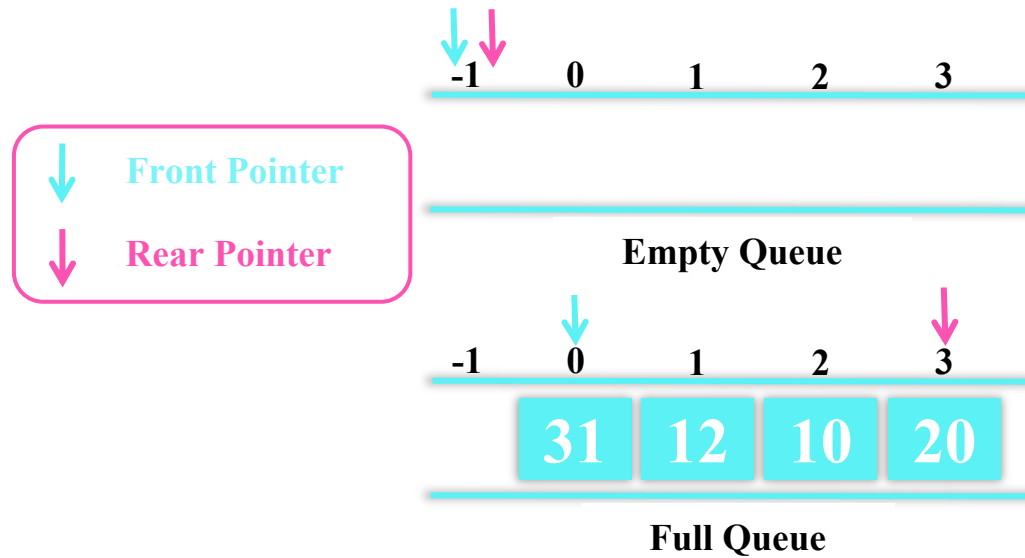
Queue

Queue



Queue

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

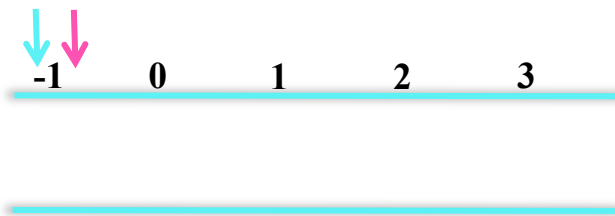


Queue

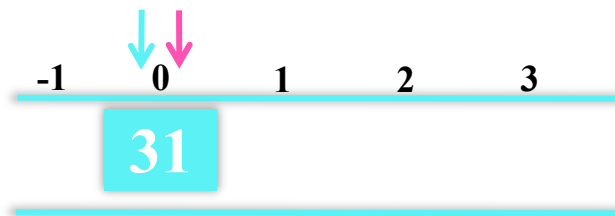


Operations

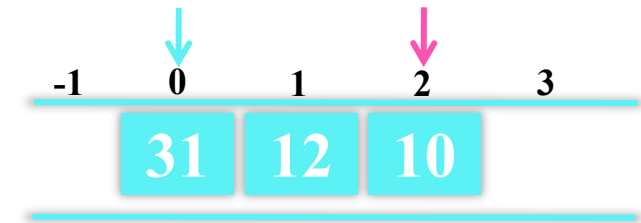
➤ Enqueue: Add an element to the end of the queue



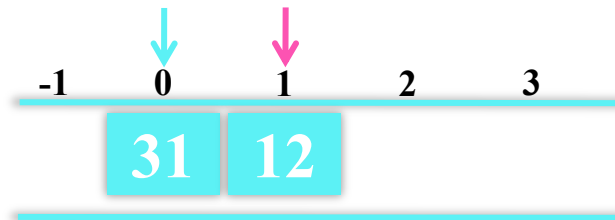
Empty Queue



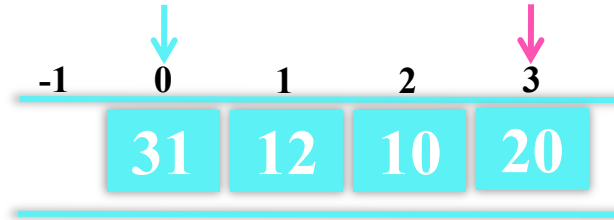
Enqueue(31)



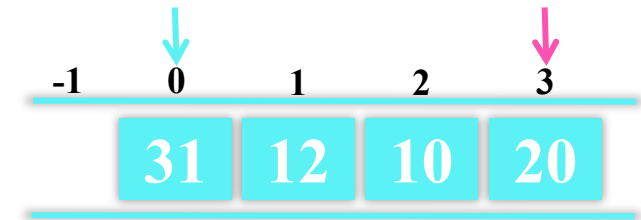
Enqueue(10)



Enqueue(12)



Enqueue(20)



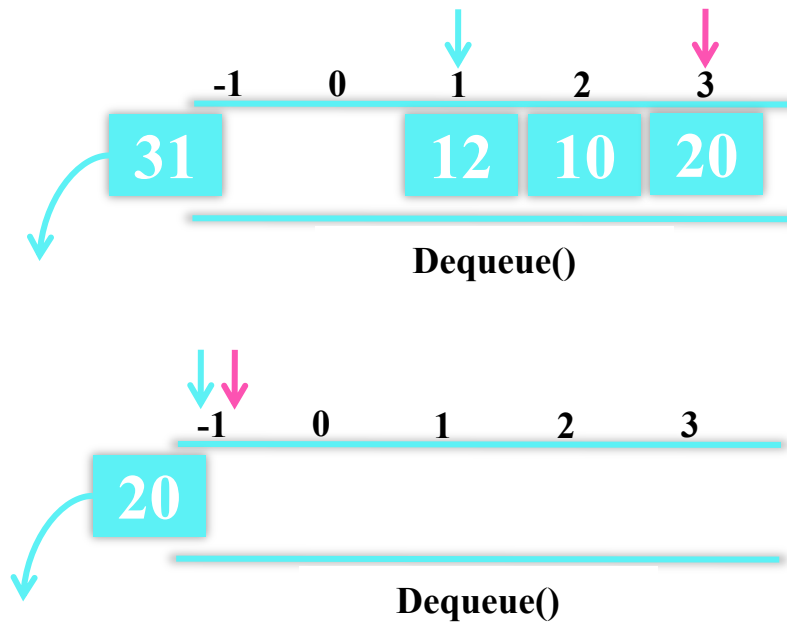
Full Queue

Queue



Operations

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

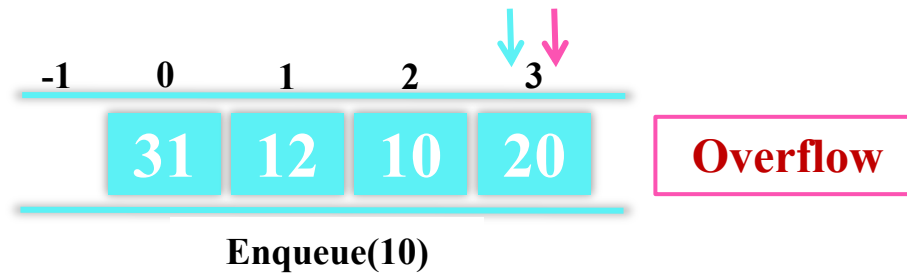


Queue



Operations

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

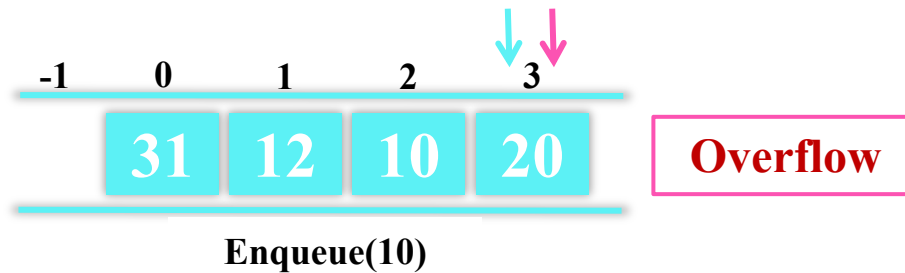


Queue



Operations

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

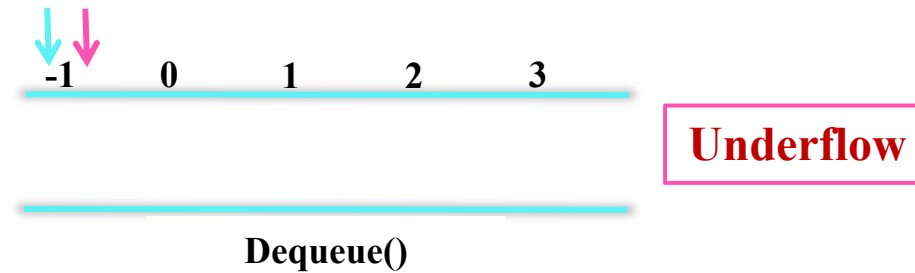


Queue



Operations

- Underflow: Try to dequeue an empty queue

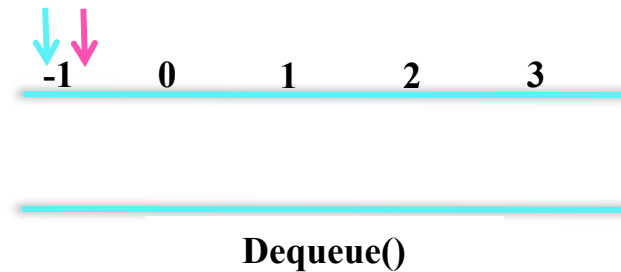


Queue



Operations

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



Underflow



Description

Queue
<ul style="list-style-type: none">- capacity: int- queue: list
<ul style="list-style-type: none">+ isEmpty(): bool+ isFull(): bool+ dequeue(): void+ enqueue(value): void+ front(): void



Solution

```
1 class MyQueue:
2     def __init__(self, capacity):
3         self.__capacity = capacity
4         self.__queue = []
5
6     def isEmpty(self):
7         return len(self.__queue) == 0
8
9     def isFull(self):
10        return len(self.__queue) == self.__capacity
11
12    def dequeue(self):
13        if self.isEmpty():
14            raise Exception("Underflow")
15        return self.__queue.pop(0)
```

```
16
17    def enqueue(self, value):
18        if self.isFull():
19            raise Exception("Overflow")
20        self.__queue.append(value)
21
22    def front(self):
23        if self.isEmpty():
24            print("Queue is empty")
25        return
26        return self.__queue[0]
```

Summary

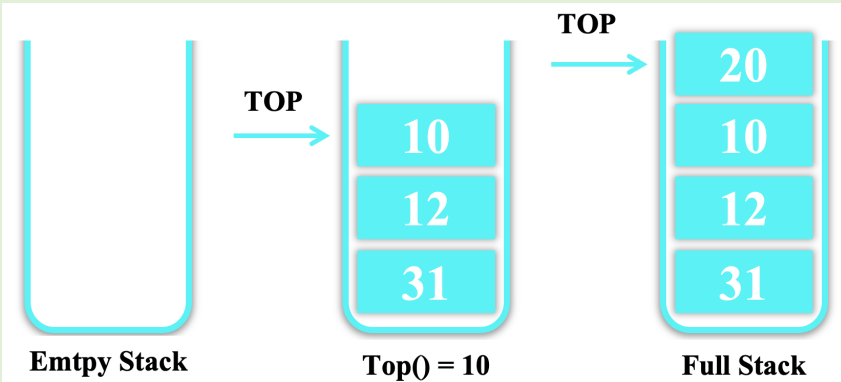
Softmax

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

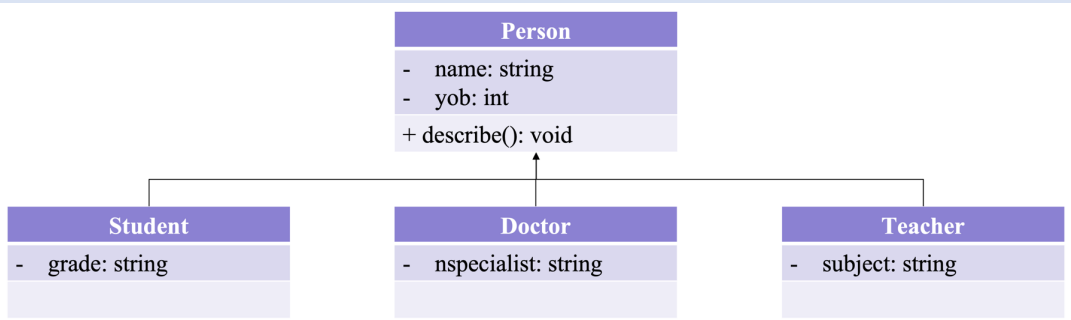
$$\text{softmax_stable}(x_i) = \frac{\exp(x_i - c)}{\sum_{j=1}^n \exp(x_j - c)}$$

$$c = \max(x)$$

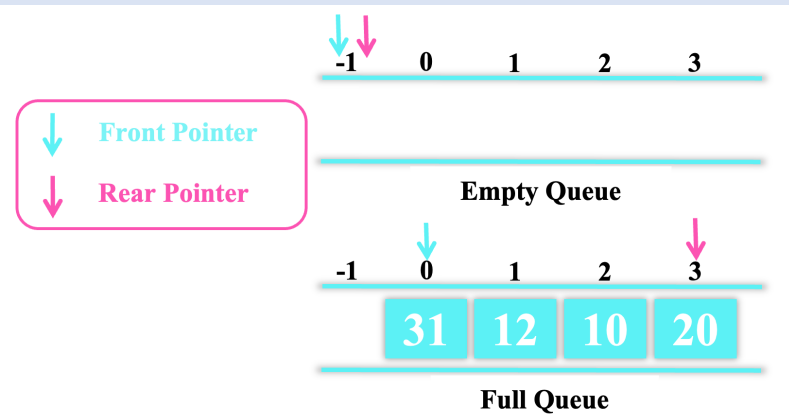
Stack



OOP (User Managemnet)



Queue





AI VIET NAM

@aivietnam.edu.vn

Thanks!

Any questions?