EXERCISE: Object Oriented Programming

AI VIET NAM
@aivietnam.edu.vn

Content

- **Review: Python Object Oriented Programming**
  - Class and Object
  - Data Encapsulation
  - Data Abstraction
  - Inheritance
  - Polymorphism
  - Aggregation and Composition

Content

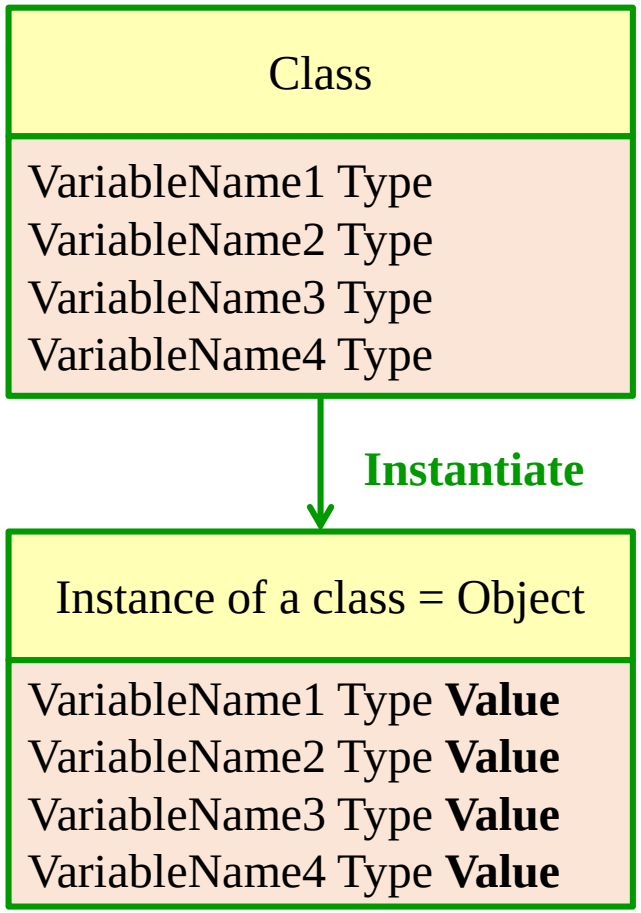
- **Exercise1: Object Oriented Aggregation**
  - Thực hiện mối quan hệ aggregation giữa 2 class Device và Maufacturer
- **Exercise2: Characteristics of Object Oriented Programming**
  - Thực hiện hệ thống cơ bản quản lý danh sách thông tin người trong một phường
- **Exercise3: Thực hiện xây dựng Sack class cơ bản**
  - Stack với các method cơ bản initialization, isEmtpy, isFull, pop, push, top
- **Exercise4: Thực hiện xây dựng Queue class cơ bản**
  - Queue với các method cơ bản initialization, isEmtpy, isFull, dequeue, enqueue, front

Content

- **Review: Python Object Oriented Programming**
  - Class and Object
  - Data Encapsulation
  - Data Abstraction
  - Inheritance
  - Polymorphism
  - Aggregation and Composition

Review: Python Object Oriented Programming

- **Class and Object**



- A **class** is a **template** for objects
- An **object** is an **instance** of a class
- Class specific what data its object will have and what behaviour will they exhibit
- Object contians its own value
- Instantiate assign value to data member (attributes)

=> A class defines what data and methods should the object have
=> object contains actual data

Review: Python Object Oriented Programming

Class and Object

Classes

Phone
Brand Name
call() takePhoto()

Person
Job Name
work() talk()

Objects

Phone
Brand - Apple Name - Iphone 13 Pro
call() takePhoto()

Phone
Brand - Samsung Name - Galaxy S21
call() takePhoto()

Person
Job - Doctor Name - Edward Jenner
work() talk()

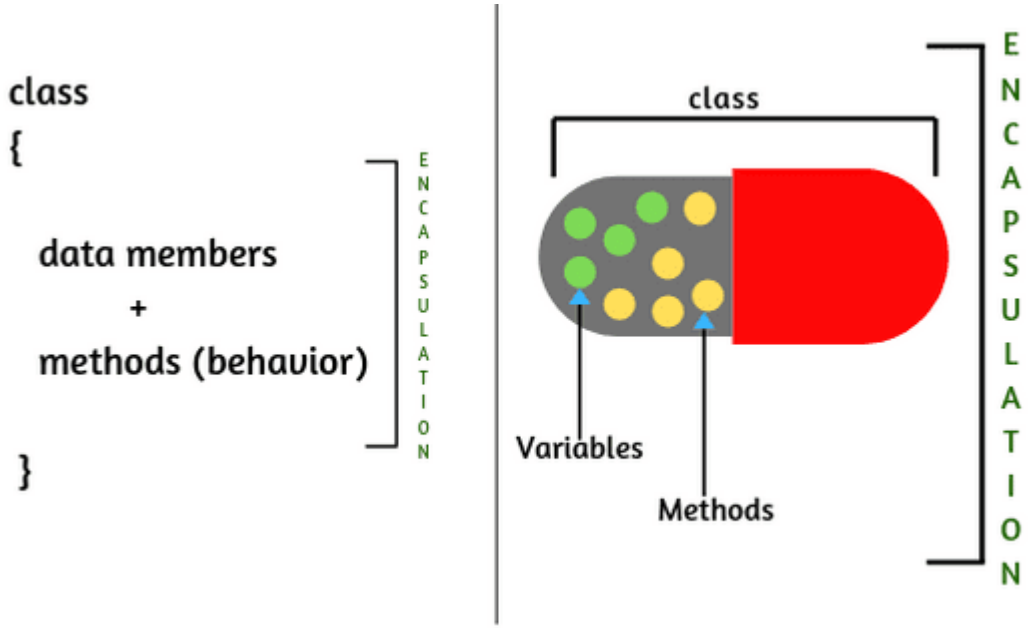
Person
Job - Teacher Name - Maria Montessori
work() talk()

Review: Python Object Oriented Programming

Data Encapsulation

Classes **encapsulate states** and **behaviour** together

+ State: is maintained through variables called **data member (attribute)**
+ Behaviour: is implemented through **methods** defined inside a class
+ Behaviour is **shared** by **all the object but data is not**




```
class Employee:
    def __init__(self, name, project):
        self.name = name
        self.project = project

    def work(self):
        print(self.name, 'is working on', self.project)
```

Data Members

Method

Wrapping data and the methods that work on data within one unit

Class (Encapsulation)

https://pynative.com/python-encapsulation/

Review: Python Object Oriented Programming

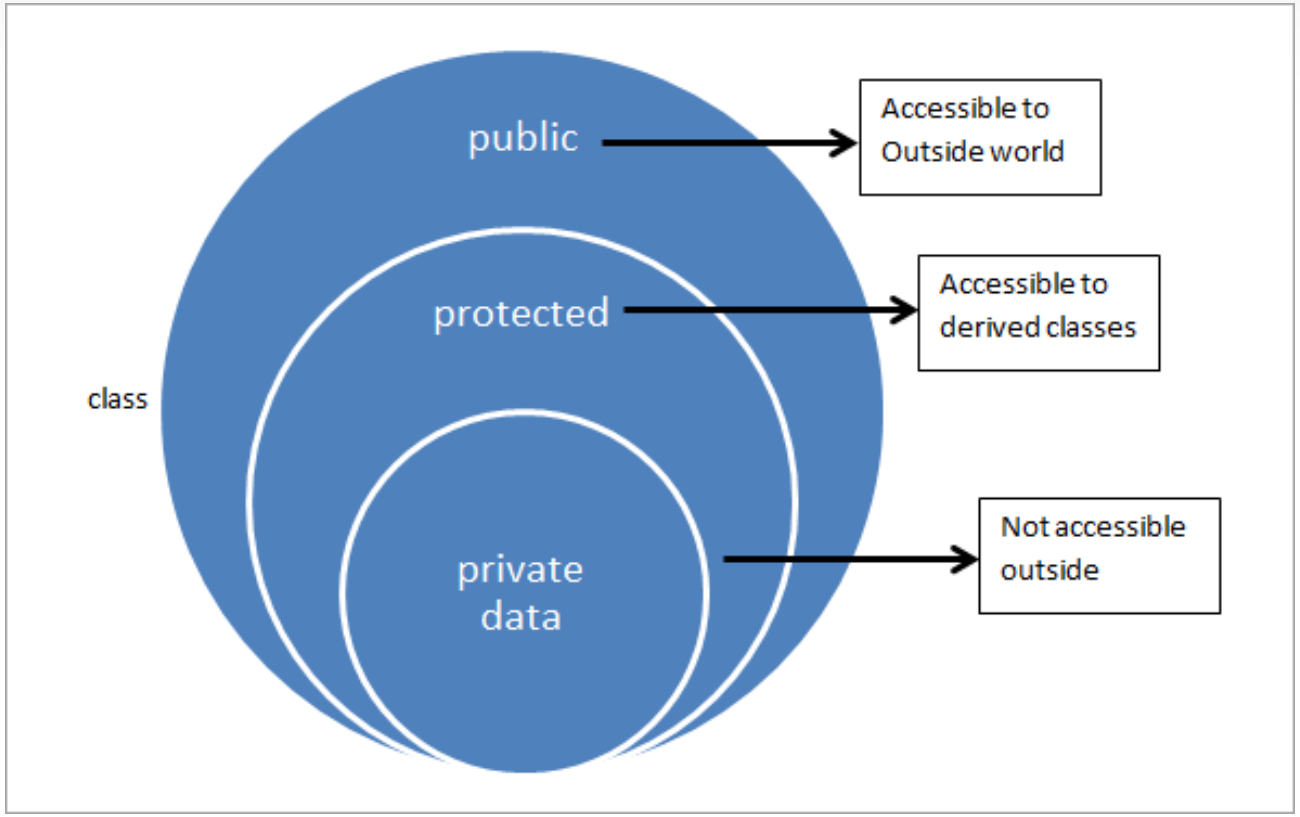
Data Encapsulation

Information hiding: hide object's internal representation from the outside

Limit Access: some data member and method cannot access from outside of class

Access modifiers help to restrcit access data members and methods of class:

- **Public member**: can be accessible anywher
- **Protected member**: can be accesible within the class and its sub-classes
- **Private member**: can be accessible within the class



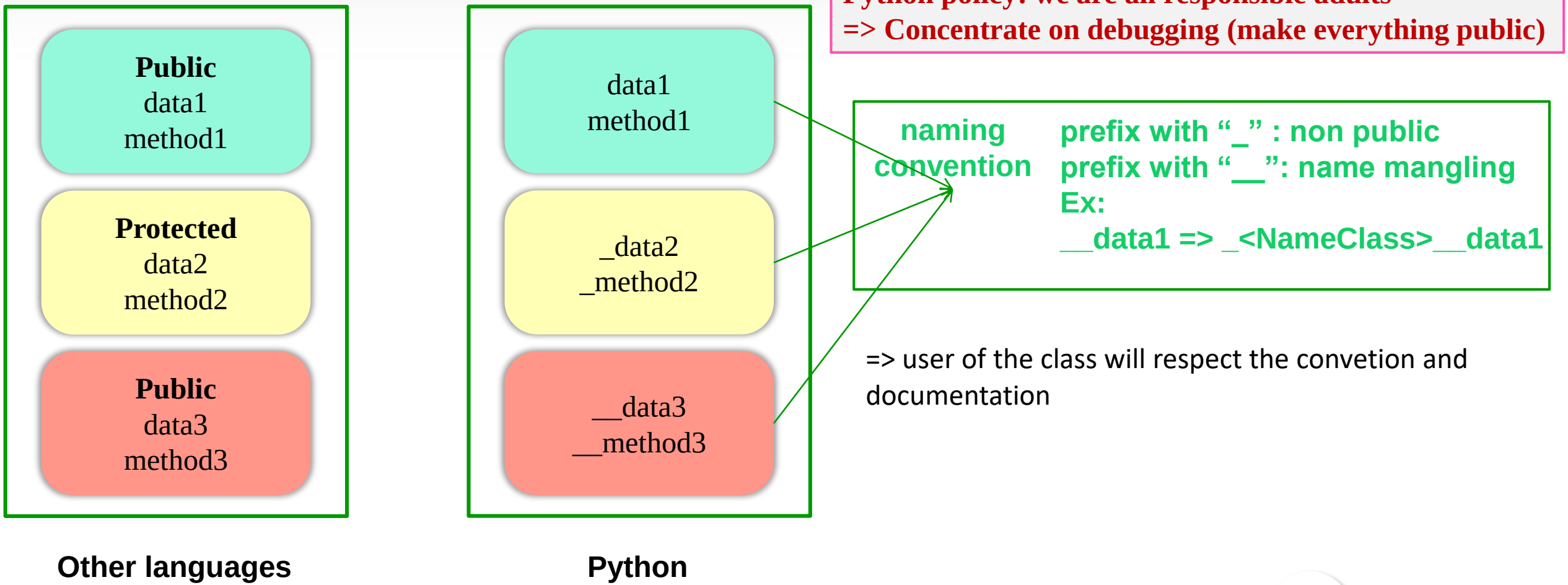
Ensure data encapsulation:

- **getter** (public method): read value of data member
- **setter** (public method): set value to data member

https://pynative.com/python-encapsulation/

Review: Python Object Oriented Programming

Data Encapsulation



Review: Python Object Oriented Programming

Data Encapsulation

Security: protect objects from unauthorized access. It provide 3 levels of access modifiers to prevent unexpected data modification

Data hiding: base on the provided getter and setter methods, it is not neccessary to know the internal running inside of object

Simplicity: It simplifies the maintenance of the application by keeping classes separated and preventing them from tightly coupling with each other

Aesthetics: Bundling data and methods within a class makes code more readable and maintainable

Review: Python Object Oriented Programming

Data Abstraction

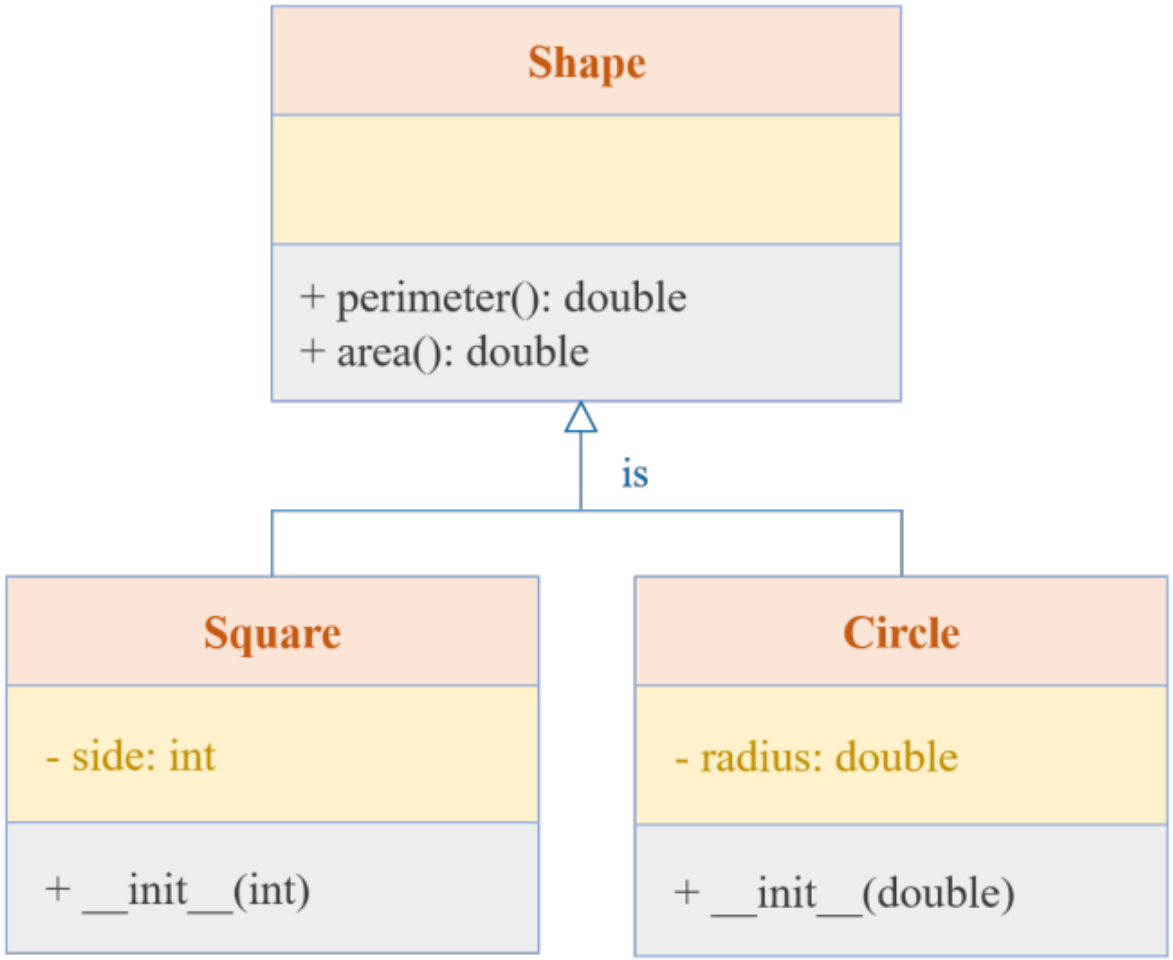
- Data abstraction refers to the process of **representing essential features** without including background details or explanations.
- Abstraction is an OOP concept that **focuses only on relevant data of an object**.
- It **hides the background details** and **emphasizes the essential data** points for reducing the complexity and increase efficiency
- Abstraction method mainly **focusses on the idea instead** of **actual functioning**

Parameter	Abstraction	Encapsulation
Use for	Abstraction solves the problem and issues that arise at the design stage.	Encapsulation solves the problem and issue that arise at the implementation stage.
Focus	Abstraction allows you to focus on what the object does instead of how it does it	Encapsulation enables you to hide the code and data into a single unit to secure the data from the outside world.
Implementation	You can use abstraction using Abstract Class.	You can implement encapsulation using Access Modifiers (Public, Protected & Private.)
Focuses	Focus mainly on what should be done.	Focus primarily on how it should be done.
Application	During design level.	During the Implementation level.

https://www.guru99.com/difference-between-abstraction-and-encapsulation.html

Review: Python Object Oriented Programming

Data Abstraction




```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def computeArea(self):
        pass
```

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

    def computeArea(self):
        return self.__side*self.__side
```

```python
square = Square(5)
print(square.computeArea())
```

x`x`

Abstraction	Encapsulation
Abstraction in Object Oriented Programming solves the issues at the design level.	Encapsulation solves it implementation level.
Abstraction in Programming is about hiding unwanted details while showing most essential information.	Encapsulation means binding the code and data into a single unit.
Data Abstraction in Java allows focussing on what the information object must contain	Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons.

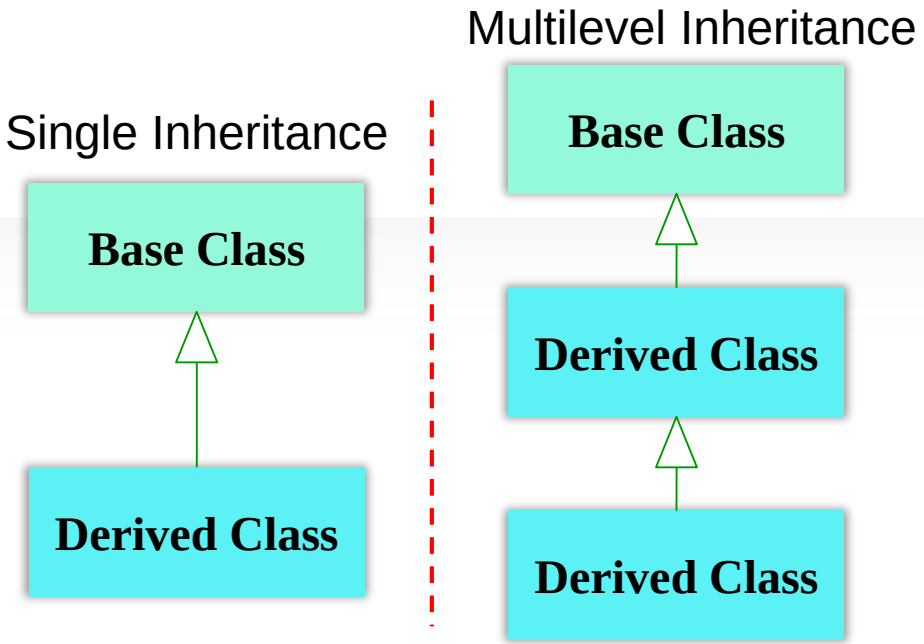
https://www.guru99.com/java-data-abstraction.html

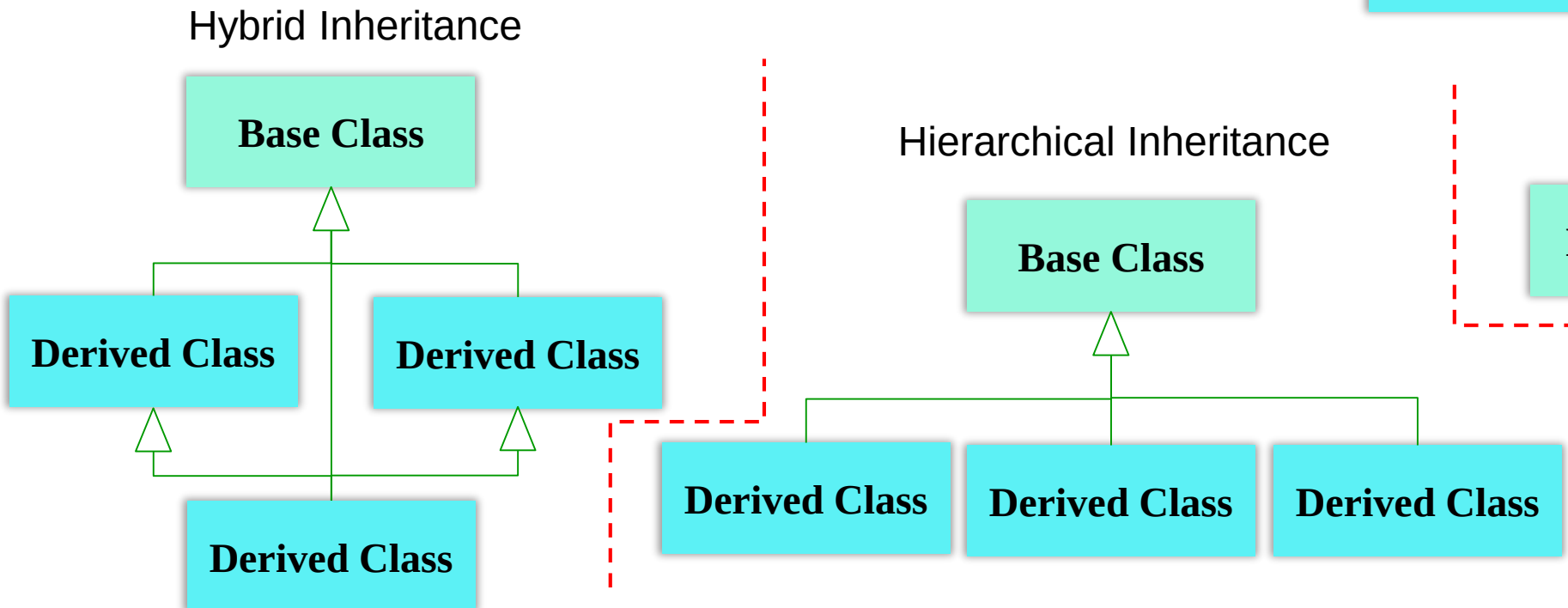
Review: Python Object Oriented Programming

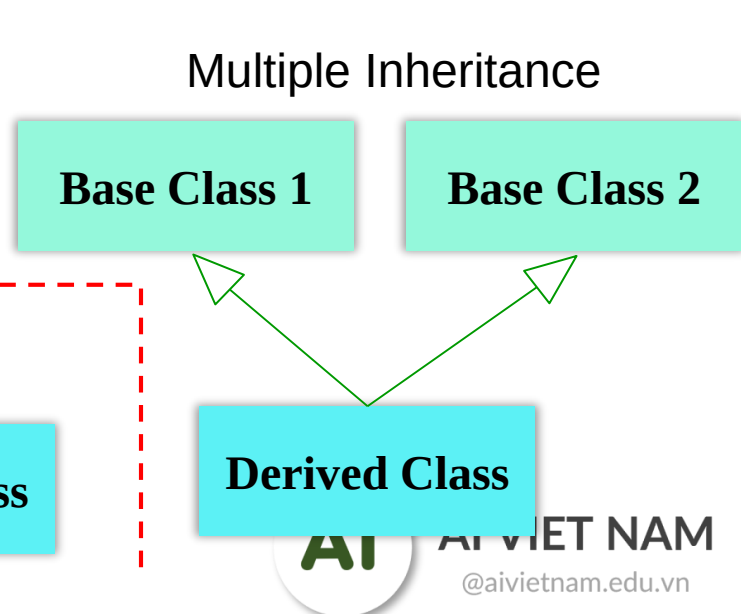
Inheritance

- **Base class** (parent): the class which is inherited from another class
- **Derived class** (child): the class inherits from another class
- **Inheritance**: the action that derived class **inherit** (public and protected) **attrtibutes and methods** from base class. Derived class can **overriding methods** of base class
- The relationship between base class and derived class is **is-a** relationship

=> **reusability of code**: use existing class to create a new class



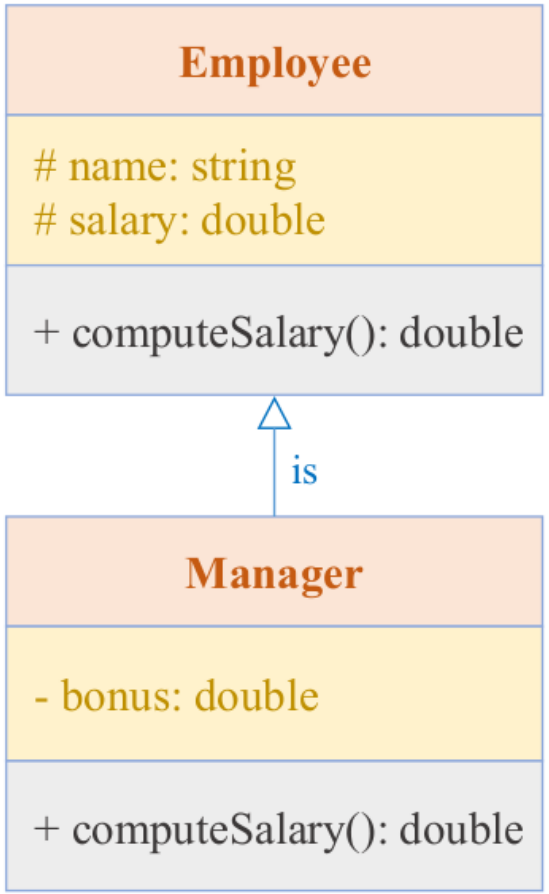




Review: Python Object Oriented Programming

Method is overrided in derived class

supper() function




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

    def computeSalary(self):
        return self._salary

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

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

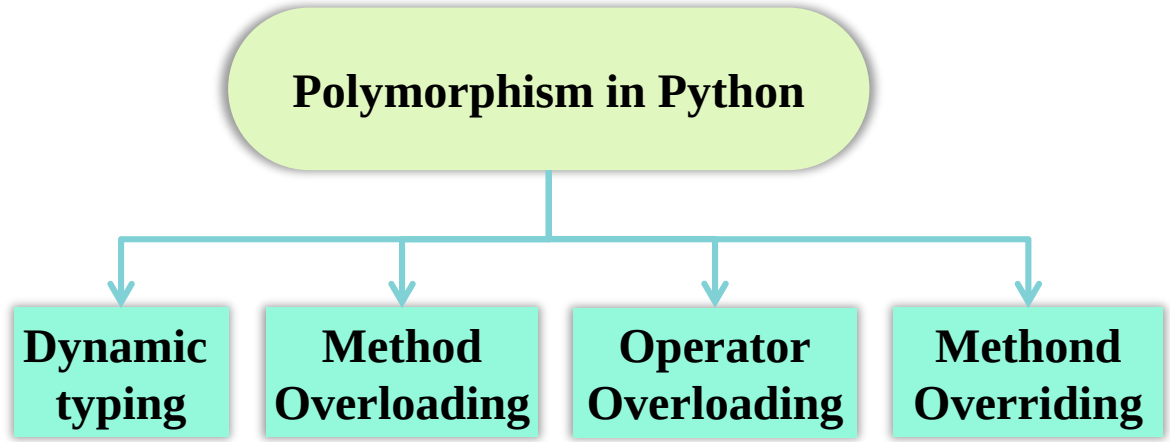
```python
peter = Manager('Peter', 100, 20)
salary = peter.computeSalary()
print(f'Peter Salary: {salary}')
```

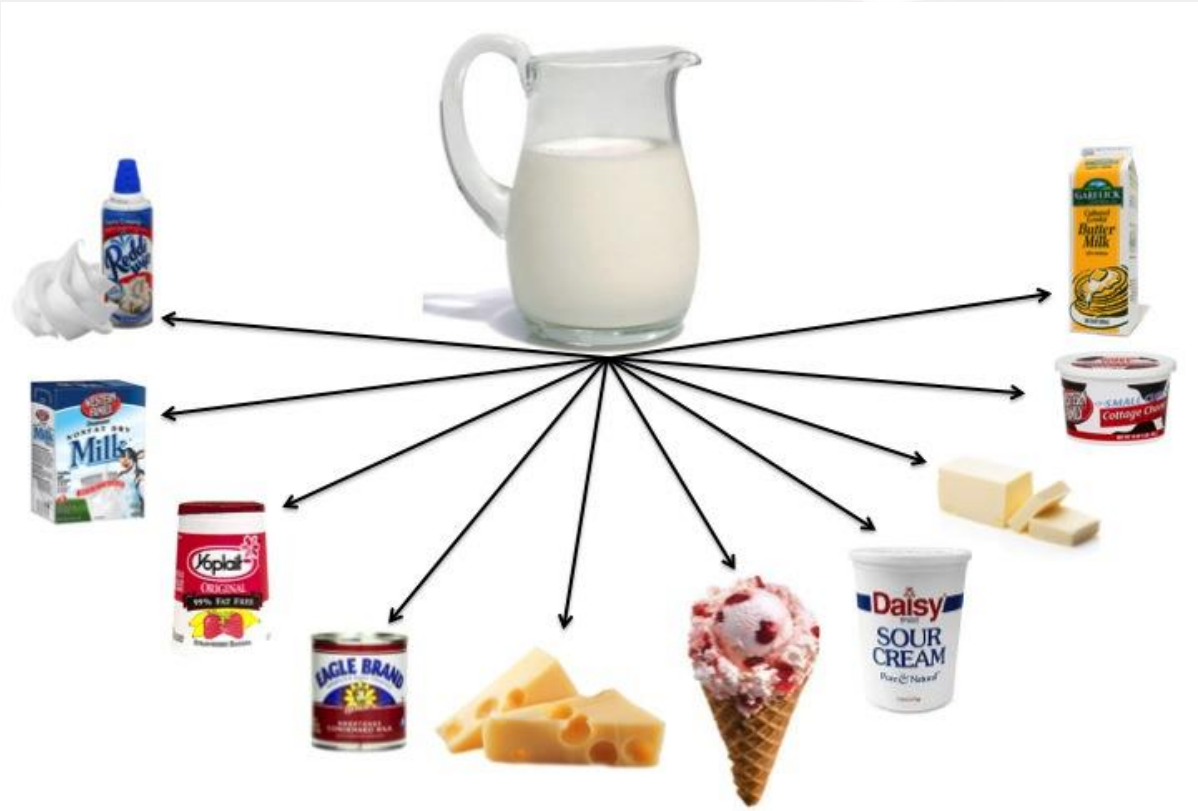
```
Peter Salary: 120
```

Review: Python Object Oriented Programming

Polymorphism

- **Poly** means **many**, **morphism** means **form**.
- **Same function** name can be **used for different types**
- Enable to use a **single interface** with the **input of different data types, different classes, or different number of inputs**

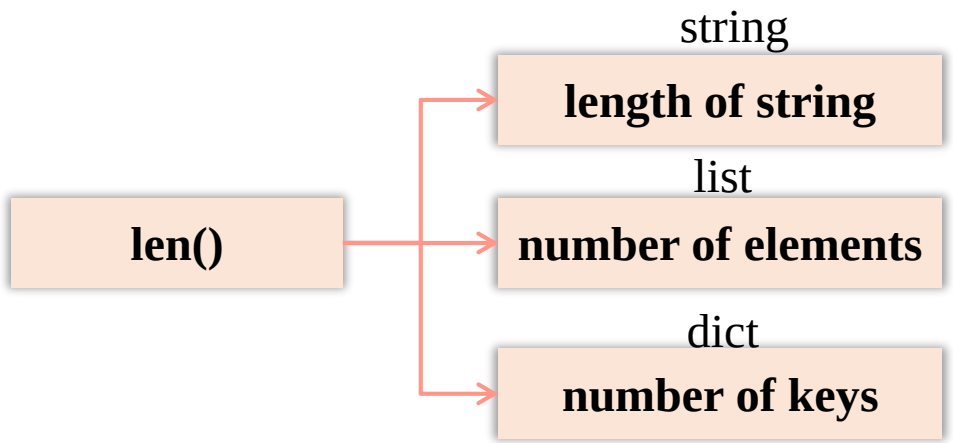




Review: Python Object Oriented Programming

Polymorphism

Dynamic typing



- Do not need to mention type of object before we perform any kind of operation on that object (at runtime)
- If a method is defined in a certain object we can invoke it at runtime

Method Overloading

```python

class Test:
    def sum(self, num1=None, num2=None, list_num=None):
        res = 0
        if (num1 is not None ) and (num2 is None) and (list_num is None):
            res = num1 + 1
        elif (num1 is not None) and (num2 is not None) and (list_num is None):
            res = num1 + num2
        elif (num1 is not None) and (num2 is None) and (list_num is None):
            for n in list_num:
                res += n
            res += num1
        else:
            raise Exception("This operation is not supported")
        return res

ins = Test()

print(ins.sum(num1=1, num2=3))
print(ins.sum(num1=2, list_num=[1,2,3,4,5]))
```

- Python does not support method overloading
- User can implement method overloading

Review: Python Object Oriented Programming

Polymorphism

Operator Overloading

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

p1 = Point(1.5, 2.5)
p2 = Point(3.1, 4.1)
p3 = p1 + p2
print(p3.x, p3.y)

```

- Provide the operators with a special meaning for a data type

Methond Overriding

```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()
```

- Derived class inherits from base class, and modifies a certain method of base class

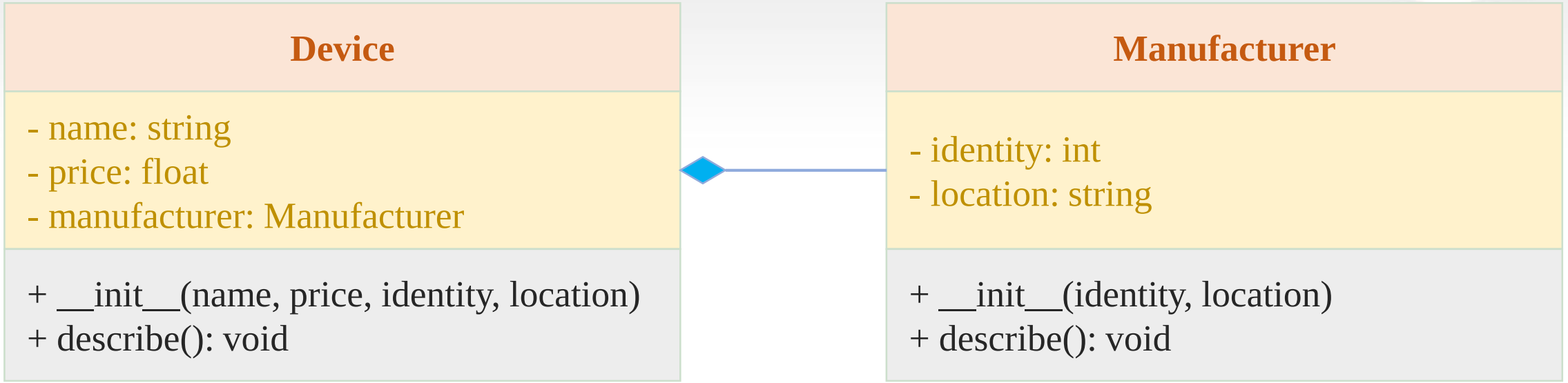
Review: Python Object Oriented Programming

Aggregation and Composition

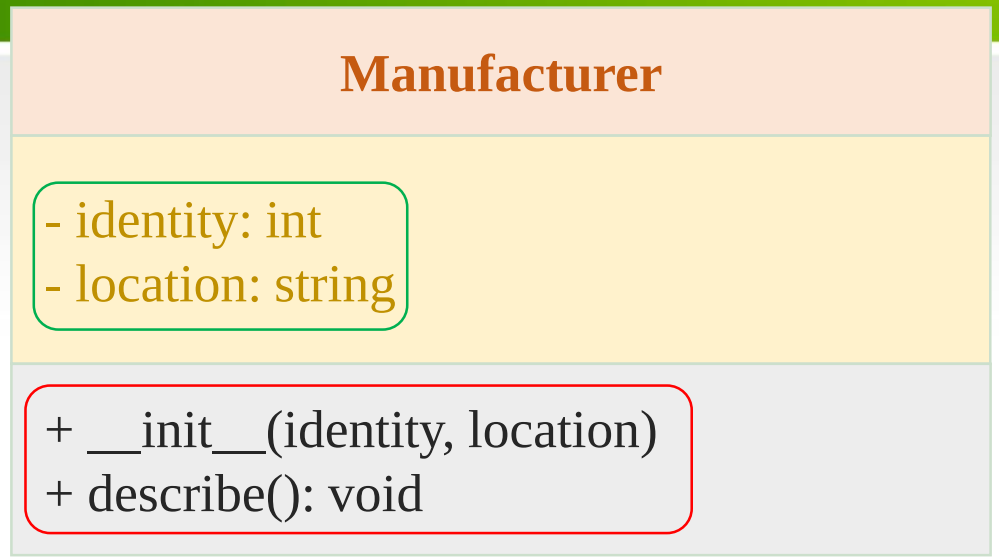
- **Composition**: One class (container) contain and use other class (content class) as a data type. If the class (container) is destroyed the content is also destroyed.
- **Aggregation**: There are not any objects or classes owns another object. It just creates a reference. If the container is destroy, the content still exists.

Content

- **Exercise1: Object Oriented Aggregation**
  - Thực hiện mối quan hệ aggregation giữa 2 class Device và Maufacturer
- **Exercise2: Characteristics of Object Oriented Programming**
  - Thực hiện hệ thống cơ bản quản lý danh sách thông tin người trong một phường
- **Exercise3: Thực hiện xây dựng Sack class cơ bản**
  - Stack với các method cơ bản initialization, isEmtpy, isFull, pop, push, top
- **Exercise4: Thực hiện xây dựng Queue class cơ bản**
  - Queue với các method cơ bản initialization, isEmtpy, isFull, dequeue, enqueue, front



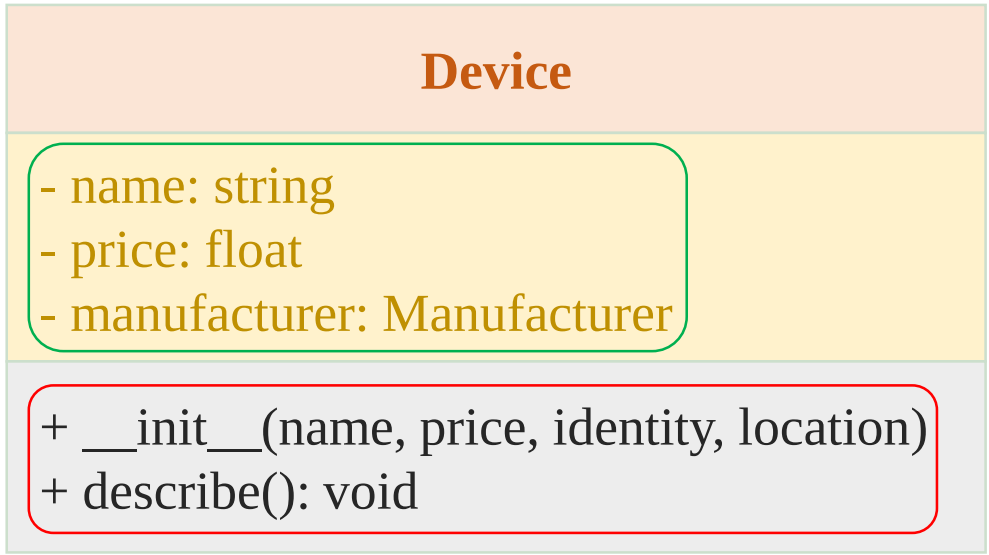

```python
# Examples 1
device1 = Device(name="mouse", price=2.5, identity=9725, location="Vietnam")
device1.describe()
>> Name: mouse - Price: 2.5
Identity: 9725 - Location: Vietnam

device2 = Device(name="monitor", price=12.5, identity=11, location="Germany")
device2.describe()
>> Name: monitor - Price: 12.5
Identity: 11 - Location: Germany
```



Manufacturer Class

- 2 private attributes
  + **identity** kiểu int
  + **location** kiểu string

- 2 public methods
  + Initialization method: nhận 2 arguments (identity và location)
  + describe() method: print các thông tin của instance (value)



Device Class

- 3 private attributes
  + **name** kiểu int
  + **price** kiểu string
  + **manufacturer** instance của Manufacturer class

- 2 public methods
  + Initialization method: nhận 4 arguments (name, price, identity và location)
  + describe() method: print các thông tin của instance (value)

Manufacturer

- identity: int
- location: string

+ __init__(identity, location)
+ describe(): void

Manufacturer Class

- 2 private attributes
  + **identity** kiểu int
  + **location** kiểu string

- 2 public methods
  + Initialization method: nhận 2 arguments (identity và location)
  + describe() method: print các thông tin của instance (value)

```
CLASS Devcie
    INIT FUNCTION __init__(identity, location)
        PRIVATE __identity = identity
        PRIVATE __location  = location
    ENDFUNCTION

    PUBLIC FUNCTION describe()
        PRINT(__identiy,  __location)
    ENDFUNCTION
ENDCLASS
```

Device

- name: string
- price: float
- manufacturer: Manufacturer

+ __init__(name, price, identity, location)
+ describe(): void

Device Class

- 3 private attributes
  + **name** kiểu int
  + **price** kiểu string
  + **manufacturer** instance của Manufacturer class

- 2 public methods
  + Initialization method: nhận 4 arguments (name, price, identity và location)
  + describe() method: print các thông tin của instance (value)

Aggregation in OOPS

```
CLASS Devcie
    INIT FUNCTION __init__(name, price, identity, location)
        PRIVATE __name = name
        PRIVATE __price = price
        PRIVATE __manufacturer = Manufacturer(identity, location)
    ENDFUNCTION

    PUBLIC FUNCTION describe()
        PRINT(__name, __price)
        __manufacturer.describe()
    ENDFUNCTION
ENDCLASS
```

Content

- **Exercise1: Object Oriented Aggregation**
  - Thực hiện mối quan hệ aggregation giữa 2 class Device và Maufacturer
- **Exercise2: Characteristics of Object Oriented Programming**
  - Thực hiện hệ thống cơ bản quản lý danh sách thông tin người trong một phường
- **Exercise3: Thực hiện xây dựng Sack class cơ bản**
  - Stack với các method cơ bản initialization, isEmtpy, isFull, pop, push, top
- **Exercise4: Thực hiện xây dựng Queue class cơ bản**
  - Queue với các method cơ bản initialization, isEmtpy, isFull, dequeue, enqueue, front

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 a list để chứa danh sách của mọi người trong Ward.

- (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.

- (b) Viết addPerson(person) method trong Ward class để add thêm một người mới với nghề nghiệp bất kỳ (student, teacher, doctor) vào danh sách người của ward. Tạo ra một ward object, và thêm vào 1 student, 2 teacher, và 2 doctor. Thực hiện describe() method để in ra tên ward (name) và toàn bộ thông tin của từng người trong ward.

- (c) Viết countDoctor() method để đếm số lượng doctor trong ward.

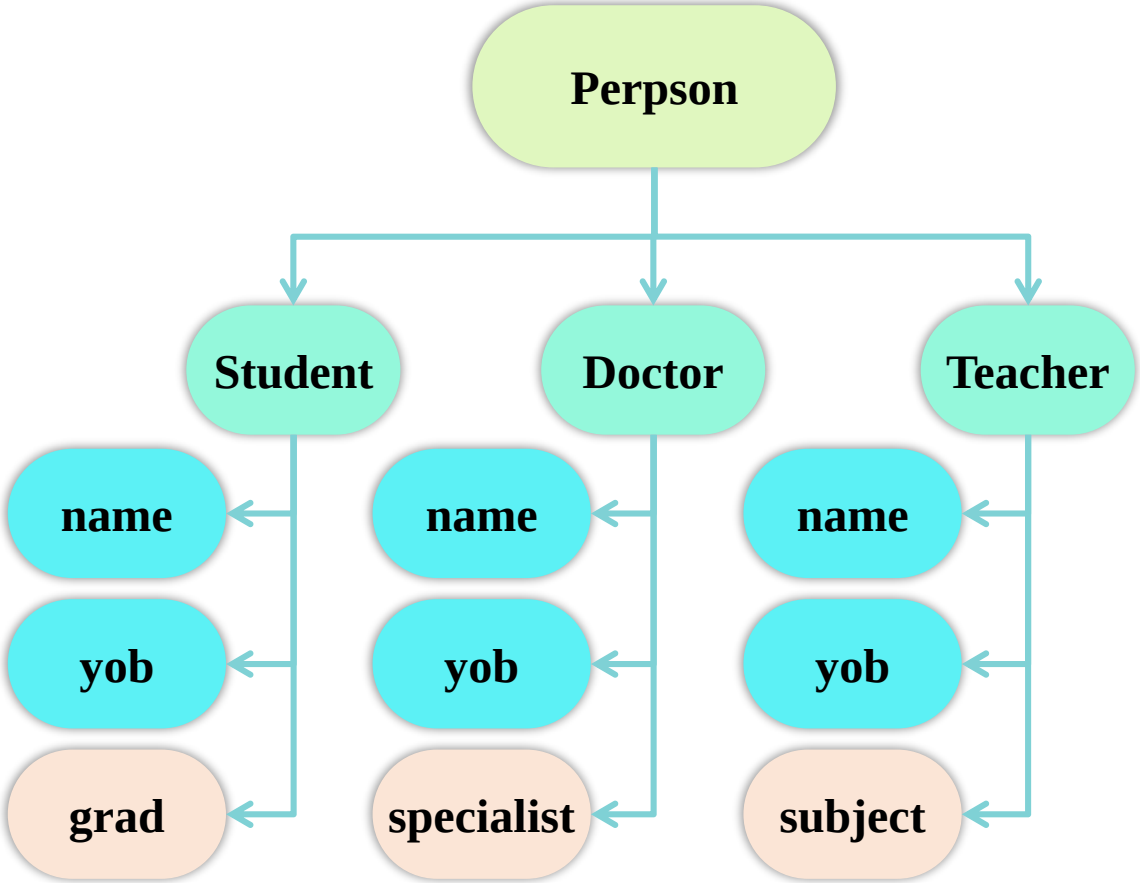
- (d) Viết sortAge() method để sort mọi người trong ward theo tuổi của họ với thứ tự tăng dần. (hint: Có thể sử dụng sort của list hoặc viết thêm function đều được)

- (e) Viết aveTeacherYearOfBirth() method để tính trung bình năm sinh của các teachers trong ward.

aa

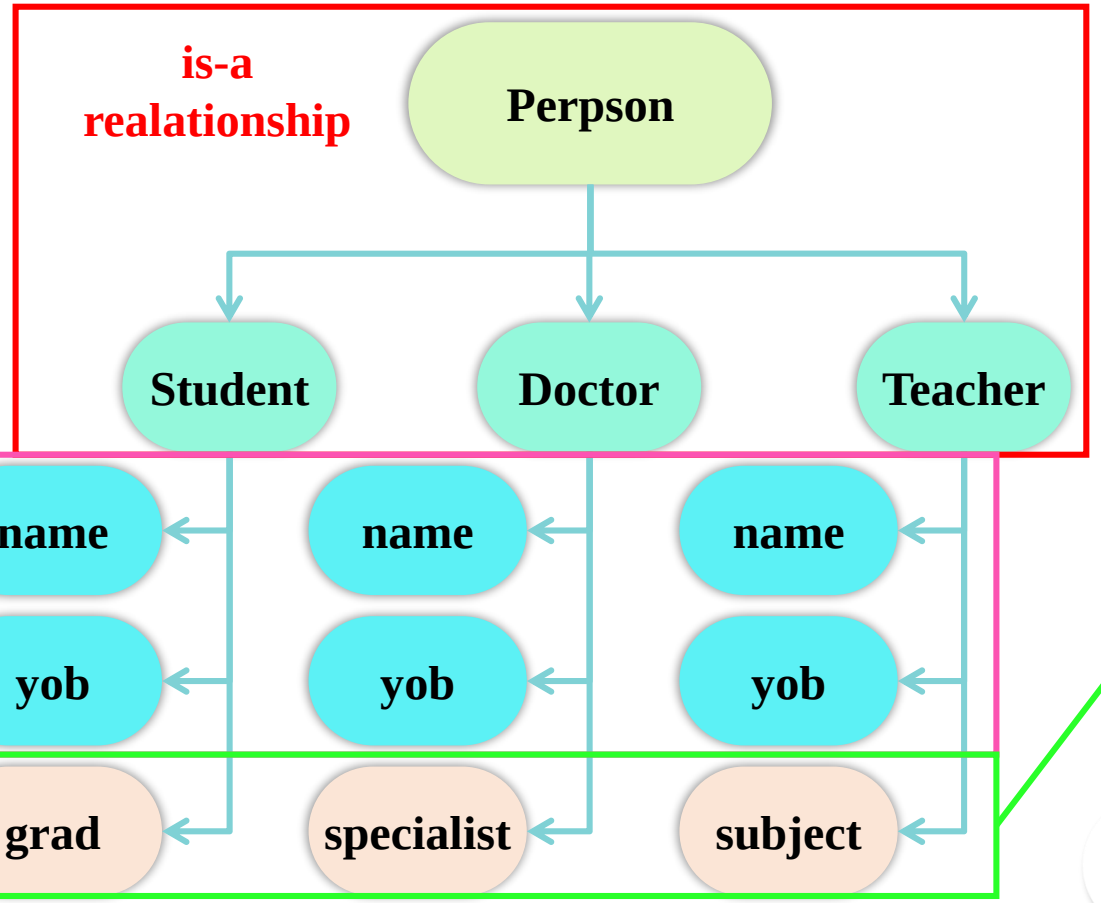
Một Ward gồm có name (string) và danh sách của mọi người trong Ward. Một người person có thể là student, doctor, hoặc teacher. Một student gồm có name, yob (int) (năm sinh), và grade (string). Một teacher gồm có name, yob, và subject (string). Một doctor gồm có name, yob, và specialist (string). Lưu ý cần sử dụng a list để chứa danh sách của mọi người trong Ward.

Ward
- name: string - listPeople: list()
+ ???



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 a list để chứa danh sách của mọi người trong Ward.

Ward
- name: string - listPeople: list()
+ ???



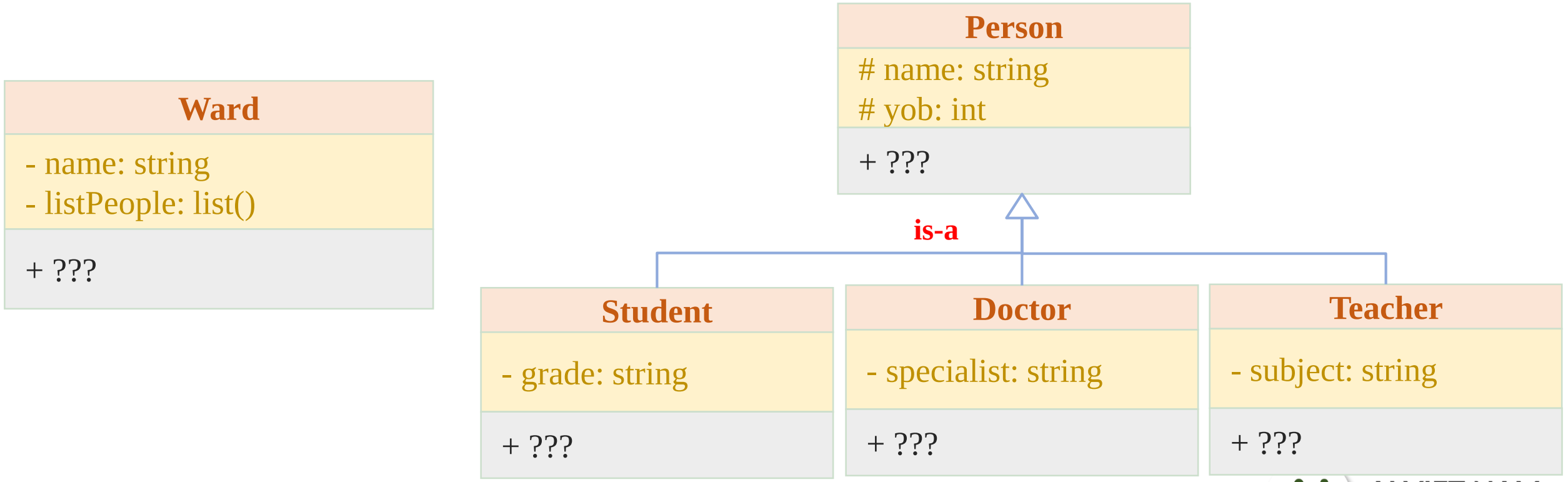
Giống nhau cho 3 class Student, Doctor và Teacher

=> Person cũng có thể có name và yob

=> Kế thừa từ Person

Đặc trưng của mỗi class

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 a list để chứa danh sách của mọi người trong Ward.



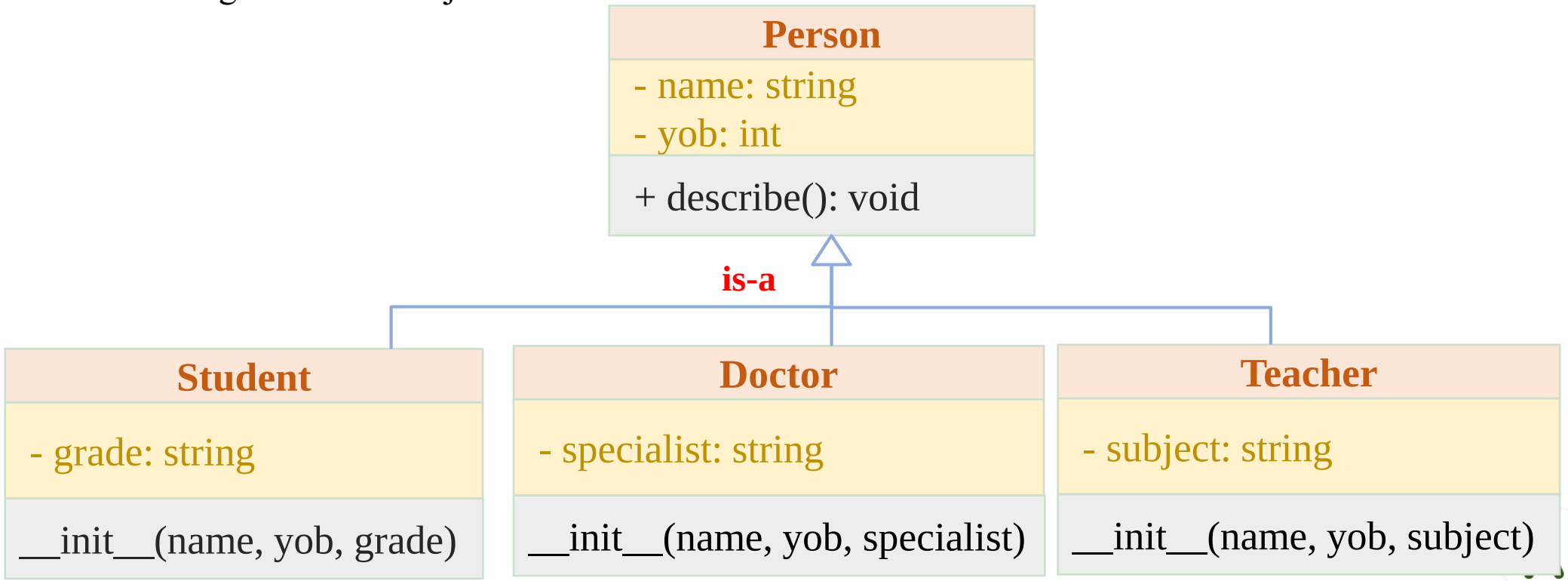
Một Ward gồm có name (string) và danh sách của mọi người trong Ward. Một người person có thể là student, doctor, hoặc teacher. Một student gồm có name, yob (int) (năm sinh), và grade (string). Một teacher gồm có name, yob, và subject (string). Một doctor gồm có name, yob, và specialist (string). Lưu ý cần sử dụng a list để chứa danh sách của mọi người trong Ward.

- (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.

```
# Examples
# 2(a)
student1 = Student(name="studentA", yob=2010, grade="7")
student1.describe()
#output
>> Student - Name: studentA - YoB: 2010 - Grade: 7

teacher1 = Teacher(name="teacherA", yob=1969, subject="Math")
teacher1.describe()
#output
>> Teacher - Name: teacherA - YoB: 1969 - Subject: Math

doctor1 = Doctor(name="doctorA", yob=1945, specialist="Endocrinologists")
doctor1.describe()
#output
>> Doctor - Name: doctorA - YoB: 1945 - Specialist: Endocrinologists
```

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 a list để chứa danh sách của mọi người trong Ward.

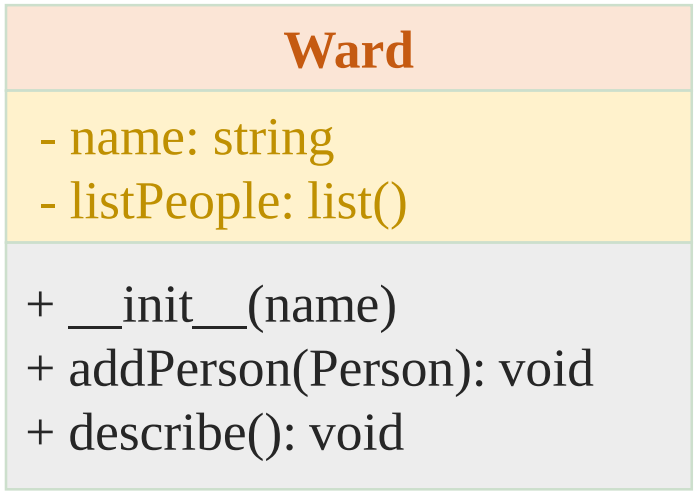
- (a) Thực hiện các class student, doctor, và teacher theo mô tả trên. Thực hiện describe() method để print ra tất cả thông tin của các objects.



- (b) Viết addPerson(person) method trong Ward class để add thêm một người mới với nghề nghiệp bất kỳ (student, teacher, doctor) vào danh sách người của ward. Tạo ra một ward object, và thêm vào 1 student, 2 teacher, và 2 doctor. Thực hiện describe() method để in ra tên ward (name) và toàn bộ thông tin của từng người trong ward.

```
# 2(b)
print()
teacher2 = Teacher(name="teacherB", yob=1995, subject="History")
doctor2 = Doctor(name="doctorB", yob=1975, specialist="Cardiologists")
ward1 = Ward(name="Ward1")
ward1.addPerson(student1)
ward1.addPerson(teacher1)
ward1.addPerson(teacher2)
ward1.addPerson(doctor1)
ward1.addPerson(doctor2)
ward1.describe()

#output
>> 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
```

- (b) Viết addPerson(person) method trong Ward class để add thêm một người mới với nghề nghiệp bất kỳ (student, teacher, doctor) vào danh sách người của ward. Tạo ra một ward object, và thêm vào 1 student, 2 teacher, và 2 doctor. Thực hiện describe() method để in ra tên ward (name) và toàn bộ thông tin của từng người trong ward.




```
CLASS Ward
    INIT FUNCTION __init__(name)
        PRIVATE __name = name
        PRIVATE __listPeople  = list()
    ENDFUNCTION

    PUBLIC FUNCTION addPerson(person)
        __listPeople.append(person)
    ENDFUNCTION

    PUBLIC FUNCTION describe()
        PRINT(__name)
        FOR p start at the first element in __listPeople TO the last element
            p.describe()
        ENDFOR
    ENDFUNCTION

ENDCLASS
```

- (c) Viết countDoctor() method để đếm số lượng doctor trong ward.

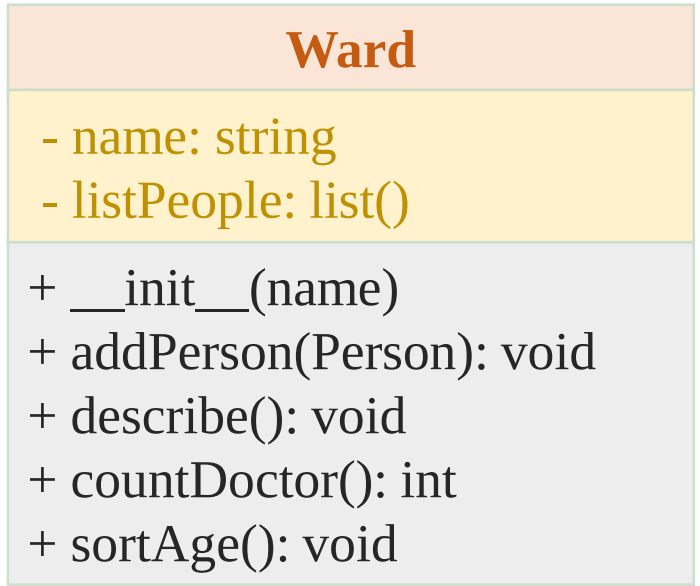
```
39 # 2(c)
40 print(f"\nNumber of doctors: {ward1.countDoctor()}")
41
42 #output
43 >> Number of doctors: 2
```

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

```
PUBLIC FUNCTION countDoctor()
    counter = 0
        FOR p start at the first element in __listPeople TO the last element
            IF p is Doctor :
                counter += 1
        ENDFOR
    RETURN counter
ENDFUNCTION
```

- (d) Viết sortAge() method để sort mọi người trong ward theo tuổi của họ với thứ tự tăng dần. (hint: Có thể sử dụng sort của list hoặc viết thêm function đều được)

```
# 2(d)
print("\nAfter sorting Age of Ward1 people")
ward1.sortAge()
ward1.describe()

#output
>> After sorting Age of Ward1 people
Ward Name: Ward1
Student - Name: studentA - YoB: 2010 - Grade: 7
Teacher - Name: teacherB - YoB: 1995 - Subject: History
Doctor - Name: doctorB - YoB: 1975 - Specialist: Cardiologists
Teacher - Name: teacherA - YoB: 1969 - Subject: Math
Doctor - Name: doctorA - YoB: 1945 - Specialist: Endocrinologists
```



- (d) Viết sortAge() method để sort mọi người trong ward theo tuổi của họ với thứ tự tăng dần. (hint: Có thể sử dụng sort của list hoặc viết thêm function đều được)

Person
name: string # yob: int
+ getYoB(): int

```
PUBLIC FUNCTION getYoB()
    RETURN _yob
ENDFUNCTION
```

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

```
PUBLIC FUNCTION sortAge()
    GET yob of each person in __listPeople
    SORT yob of each person in __listPeople in descending order
ENDFUNCTION
```

- (e) Viết aveTeacherYearOfBirth() method để tính trung bình năm sinh của các teachers trong ward.

Person
name: string # yob: int
+ getYoB(): int

```
59 # 2(e)
60 print(f"\nAverage year of birth (teachers): {ward1.aveTeacherYearOfBirth()}")
61
62 #output
63 >> Average year of birth (teachers): 1982.0
```

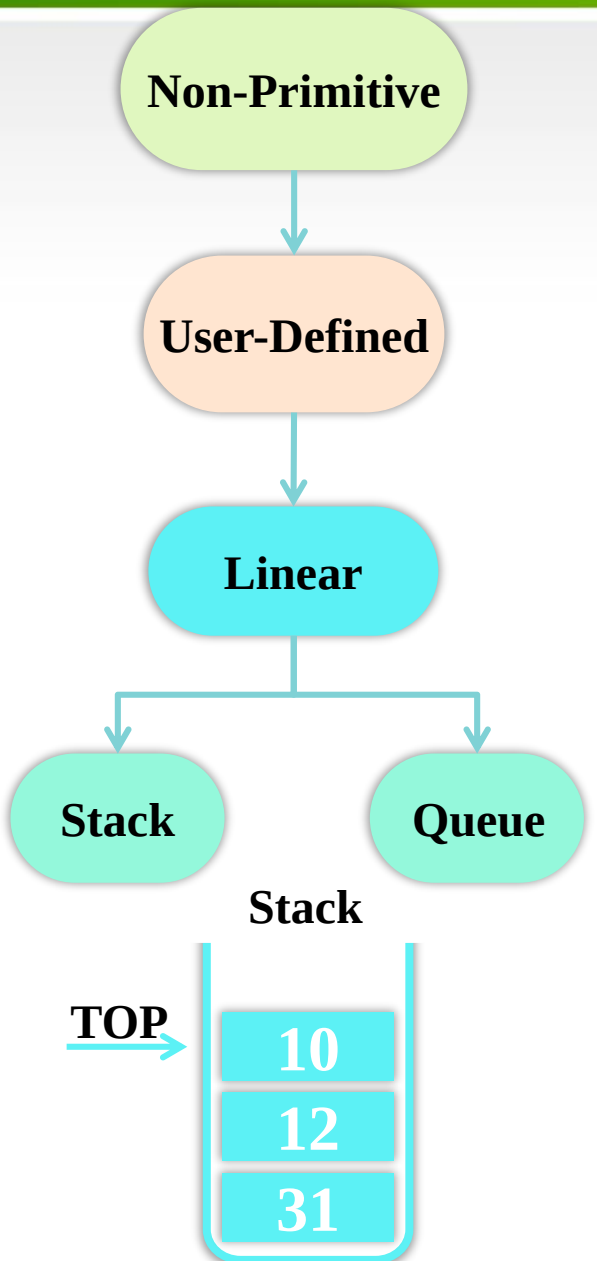
Ward
- name: string - listPeople: list()
+ __init__(name) + addPerson(Person): void + describe(): void + countDoctor(): int + sortAge(): void + aveTeacherYearOfBirth(): float

```
PUBLIC FUNCTION aveTeacherYearOfBirth()
    counter = 0
    total_year = 0
        FOR p start at the first element in __listPeople TO the last element
            IF p is Teacher:
                counter += 1
                total_year += p.getYoB()
        ENDFOR
    RETURN total_year/counter
ENDFUNCTION
```

Content

- **Exercise1: Object Oriented Aggregation**
  - Thực hiện mối quan hệ aggregation giữa 2 class Device và Maufacturer
- **Exercise2: Characteristics of Object Oriented Programming**
  - Thực hiện hệ thống cơ bản quản lý danh sách thông tin người trong một phường
- **Exercise3: Thực hiện xây dựng Sack class cơ bản**
  - Stack với các method cơ bản initialization, isEmtpy, isFull, pop, push, top
- **Exercise4: Thực hiện xây dựng Queue class cơ bản**
  - Queue với các method cơ bản initialization, isEmtpy, isFull, dequeue, enqueue, front

Review: Python Data Structures



Stack:
- **Pre-defined capacity**, can store the elements of a limited size.
- An **ordered** list (**order** in which the **data arrives** is **important**)
- **Insertion** and **Deletion** are done **at one end** called **top**.
- Top pointer (**top**) pointing to the topmost element of the stack
- The **last element inserted** is the **first one to be deleted**. Last in First out (**LIFO**) principle

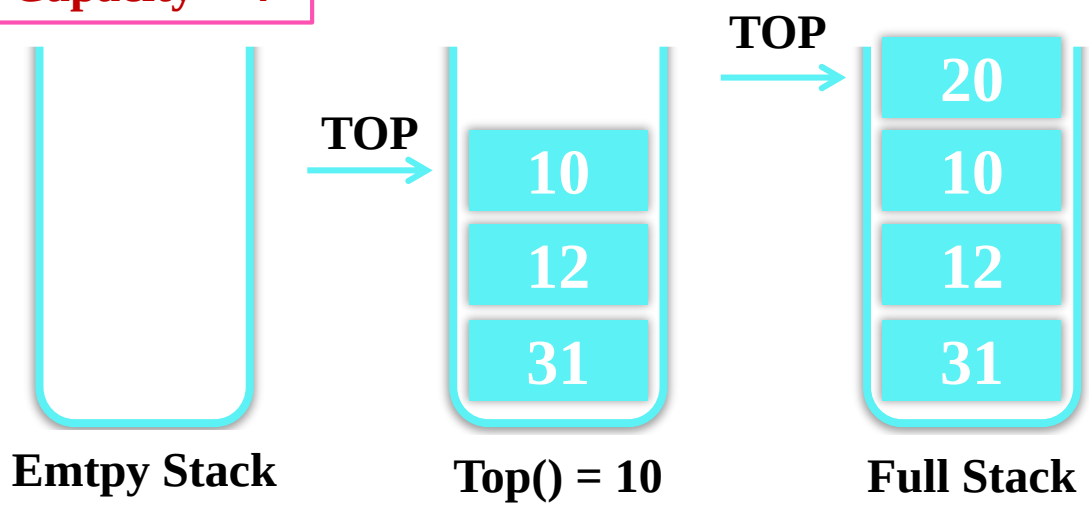
Stack Operations:
- **Push(**value**)**: insert data onto stack
- **Pop()**: remove and return the last inserted element from the stack

Auxiliary stack operations:
- **Top()**: return the last inserted element (top) without removing it
- **IsEmpty()**: indicate whether the stack is empty or not
- **IsFull()**: indicate whether the stack is full or not

Stack Exceptions:
- **Overflow**: Try to push an element to a full stack
- **Underflow**: Try to pop out an empty stack

Review: Python Data Structures

Capacity = 4

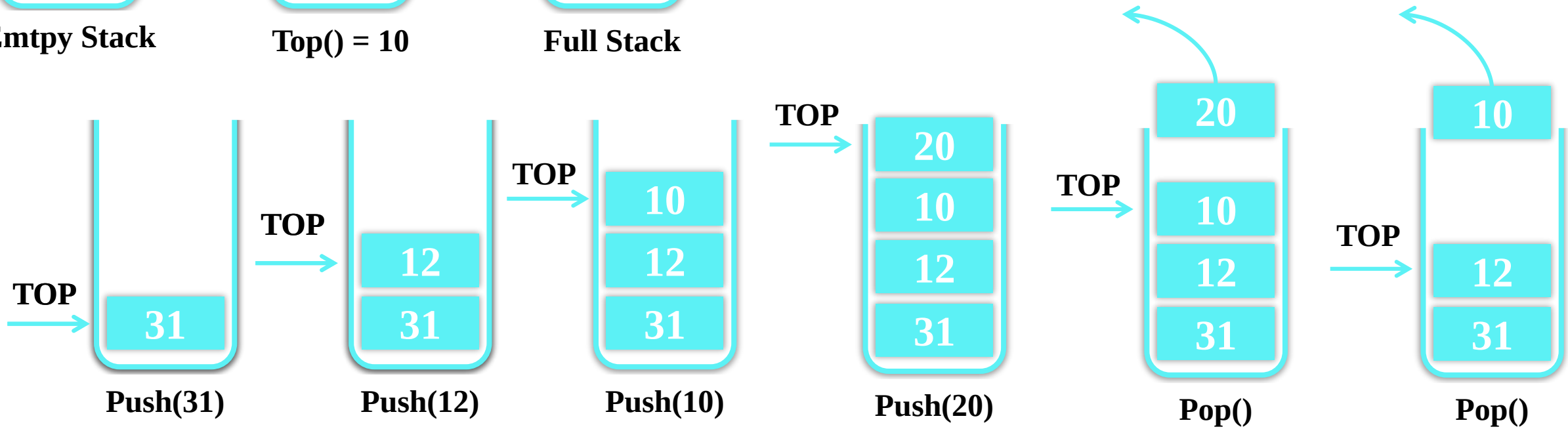


Stack Operations:

- **Push(**value**)**: insert data onto stack
- **Pop()**: remove and return the last inserted element from the stack

Auxiliary stack operations:

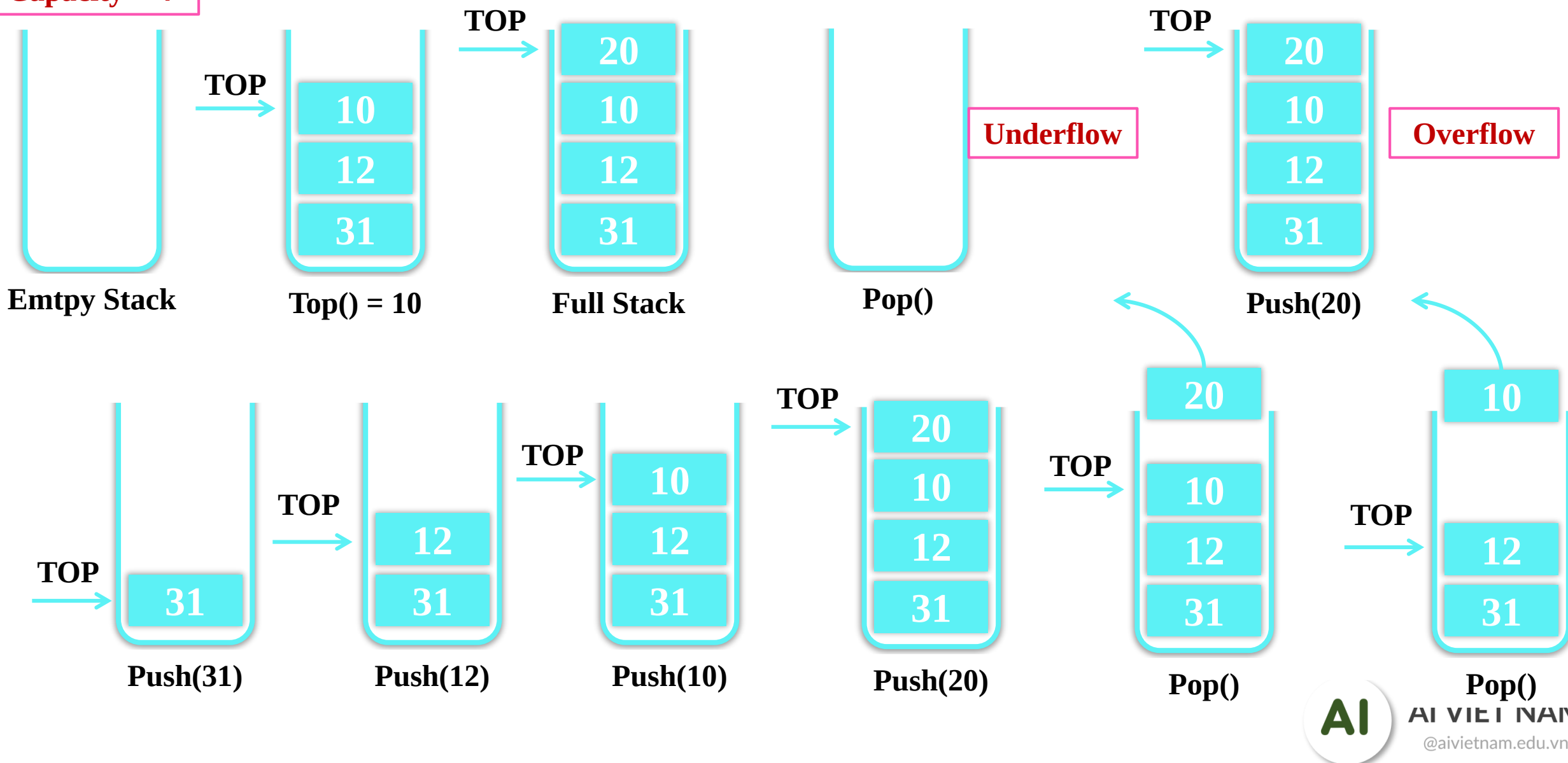
- **Top()**: return the last inserted element (top) without removing it
- **IsEmp()**: indicate whether the stack is empty or not
- **IsFull()**: indicate whether the stack is full or not



Review: Python Data Structures

Stack Exceptions:
- **Overflow**: Try to push an element to a full stack
- **Underflow**: Try to pop out an empty stack

Capacity = 4



Thực hiện xây dựng class Stack với các chức năng (method) sau đây

- **initialization method** nhận một input "capacity": dùng để khởi tạo stack với capacity là số lượng element mà stack có thể chứa
- .**isEmpty**(): kiểm tra stack có đang rỗng
- .**isFull**(): kiểm tra stack đã full chưa
- .**pop**(): loại bỏ top element và trả về giá trị đó
- .**push**(value) add thêm value vào trong stack
- .**top**() lấy giá trị top element hiện tại của stack, nhưng không loại bỏ giá trị đó
- Không cần thiết phải thực hiện với pointer như trong hình minh họa

```
stack1 = MyStack(capacity=5)

stack1.push(1)

stack1.push(2)

print(stack1.isFull())
>> False

print(stack1.top())
>>2

print(stack1.pop())
>> 2

print(stack1.top())
>> 1

print(stack1.pop())
>> 1

print(stack1.isEmpty())
>> True
```

Stack
- capacity: int - stack: list()
+ __init__(capacity) + isEmpty(): bool + isFull(): bool + pop(): any + push(value): void + top(): any

Thực hiện xây dựng class Stack với các chức năng (method) sau đây

- **initialization method** nhận một input "capacity": dùng để khởi tạo stack với capacity là số lượng element mà stack có thể chứa
- .**isEmpty**(): kiểm tra stack có đang rỗng
- .**isFull**(): kiểm tra stack đã full chưa
- .**pop**(): loại bỏ top element và trả về giá trị đó
- .**push**(value) add thêm value vào trong stack
- .**top**() lấy giá trị top element hiện tại của stack, nhưng không loại bỏ giá trị đó
- Không cần thiết phải thực hiện với pointer như trong hình minh họa

Stack

- capacity: int
- stack: list()

+ __init__(capacity)
+ isEmpty(): bool
+ isFull(): bool
+ pop(): any
+ push(value): void
+ top(): any

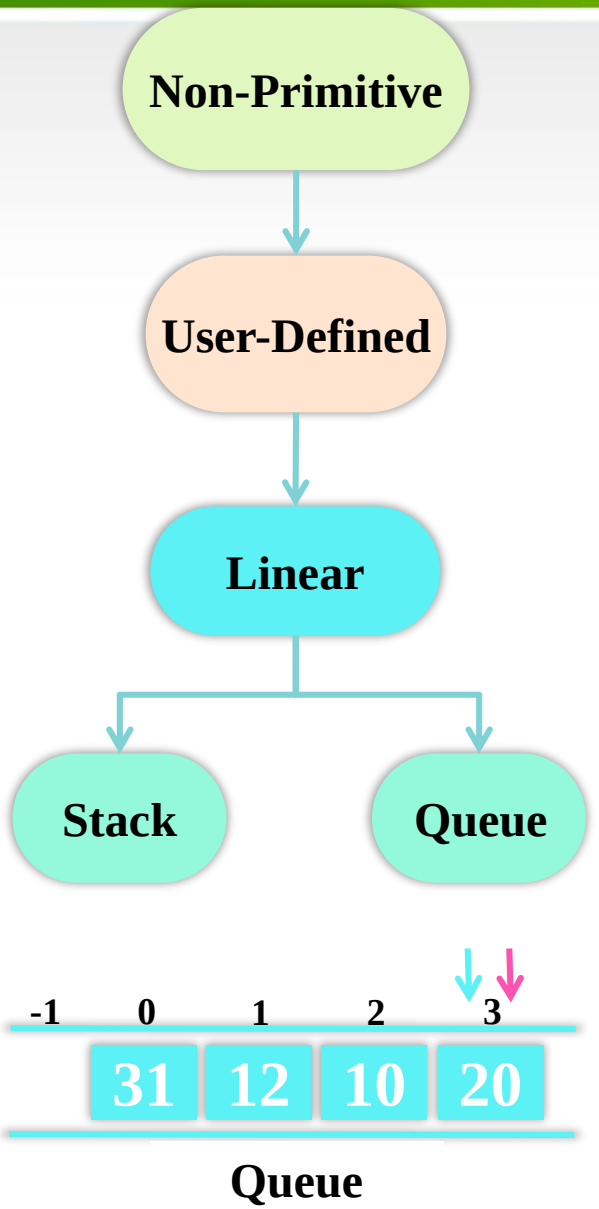
```
CLASS Stack
    INIT FUNCTION __init__(capacity)
        PRIVATE __capacity = capacity
        PRIVATE __stack = list()
    ENDFUNCTION

    PUBLIC FUNCTION isEmpty()
        RETURN len(__stack) == 0
    ENDFUNCTION

    PUBLIC FUNCTION isFull()
        RETURN len(__stack) == __capacity
    ENDFUNCTION

    PUBLIC FUNCTION pop()
        IF isEmpty()
            RAISE EXCEPTION("Underflow")
        ENDIF
        RETURN __stack.pop()
    ENDFUNCTION
...
```

```
    PUBLIC FUNCTION push(value)
        IF isFull()
            RAISE EXCEPTION("Overflow")
        ENDIF
        __stack.append(value)
    ENDFUNCTION

    PUBLIC top()
        IF isEmpty()
            PRINT("Stack is empty")
            RETURN
        ENDIF
        RETURN __stack.[-1]
    ENDFUNCTION

ENDCLASS
```

Content

- **Exercise1: Object Oriented Aggregation**
  - Thực hiện mối quan hệ aggregation giữa 2 class Device và Maufacturer
- **Exercise2: Characteristics of Object Oriented Programming**
  - Thực hiện hệ thống cơ bản quản lý danh sách thông tin người trong một phường
- **Exercise3: Thực hiện xây dựng Sack class cơ bản**
  - Stack với các method cơ bản initialization, isEmtpy, isFull, pop, push, top
- **Exercise4: Thực hiện xây dựng Queue class cơ bản**
  - Queue với các method cơ bản initialization, isEmtpy, isFull, dequeue, enqueue, front

Review: Python Data Structures



Queue:
- **Pre-defined capacity**, can store the elements of a limited size.
- An **ordered** list (**order** in which the **data arrives** is **important**)
- **Insertions** are done at one end (**rear**)
- **Deletions** are done at other end (**front**)
- Rear pointer (**rear**) pointing to the **last** element of the queue
- Front pointer (**front**) pointing to the **first** element of the queue
- The **first element inserted** is the **first one to be deleted**. Frist in First out (**FIFO**) principle

Queue Operations:
- **Enqueue(**value**)**: insert data onto queue
- **Dequeue()**: remove and return the first inserted element from the queue

Auxiliary stack operations:
- **Front()**: return the first inserted element (front) without removing it
- **IsEmptyQueue()**: indicate whether the queue is empty or not
- **IsFullQueue()**: indicate whether the queue is full or not

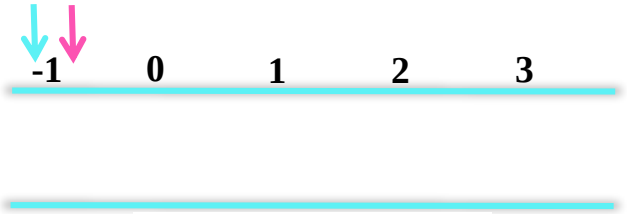
Queue Exceptions:
- **Overflow**: Try to enqueue an element to a full queue
- **Underflow**: Try to dequeue an empty queue

Review: Python Data Structures

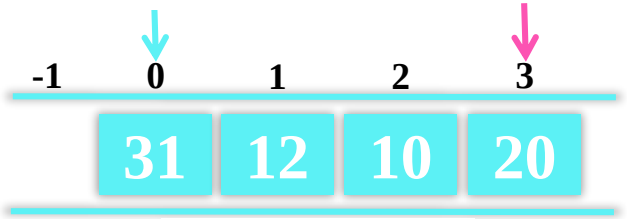
Capacity = 4

Front Pointer

Rear Pointer



Empty Queue



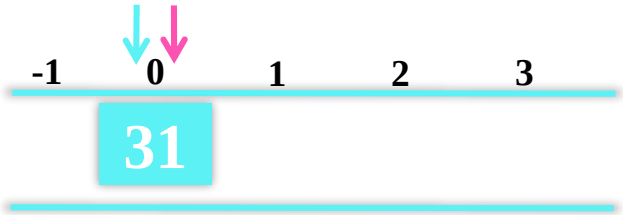
Full Queue

Queue Operations:

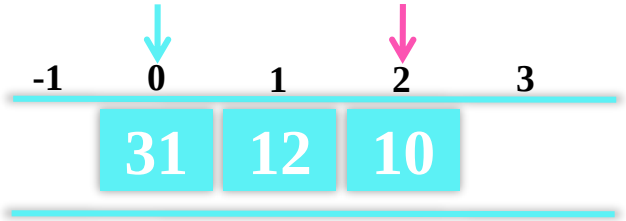
- **Enqueue(**value**)**: insert data onto queue
- **Dequeue()**: remove and return the first inserted element from the queue

Auxiliary stack operations:

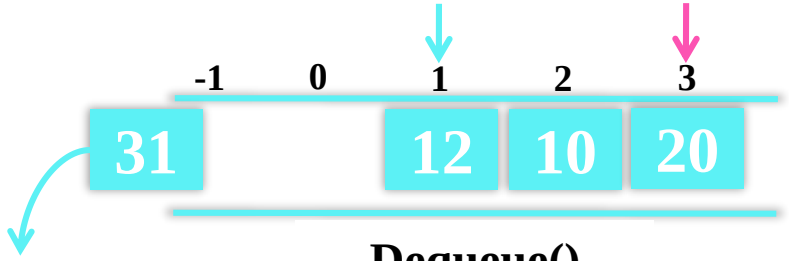
- **Front()**: return the first inserted element (front) without removing it
- **IsEmptyQueue()**: indicate whether the queue is empty or not
- **IsFullQueue()**: indicate whether the queue is full or not



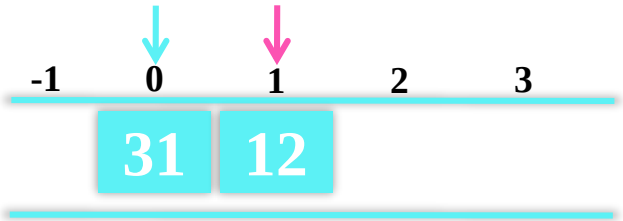
Enqueue(31)



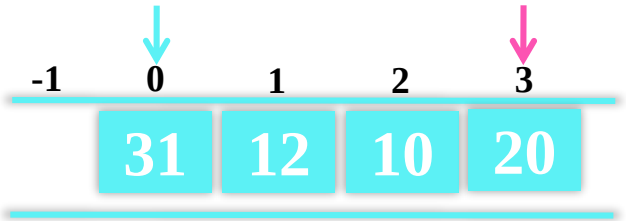
Enqueue(10)



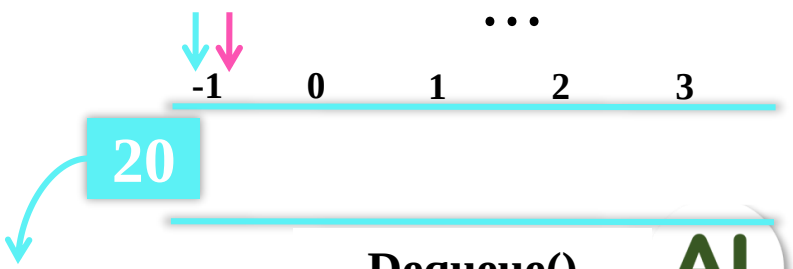
Dequeue()



Enqueue(12)



Enqueue(20)



Dequeue()

Review: Python Data Structures

Queue Exceptions:
- **Overflow**: Try to enqueue an element to a full queue
- **Underflow**: Try to dequeue an empty queue

Capacity = 4

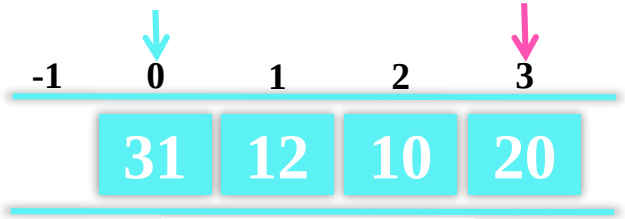
Front Pointer

Rear Pointer

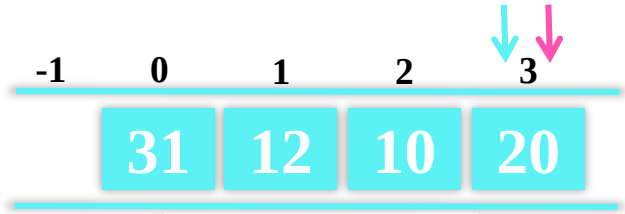
Empty Queue

Dequeue()

Underflow

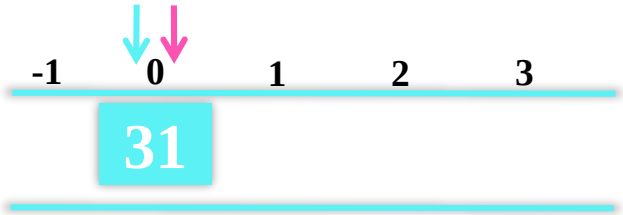


Full Queue

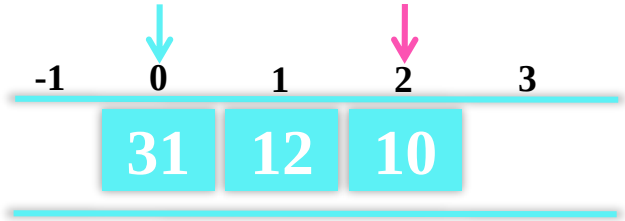


Enqueue(10)

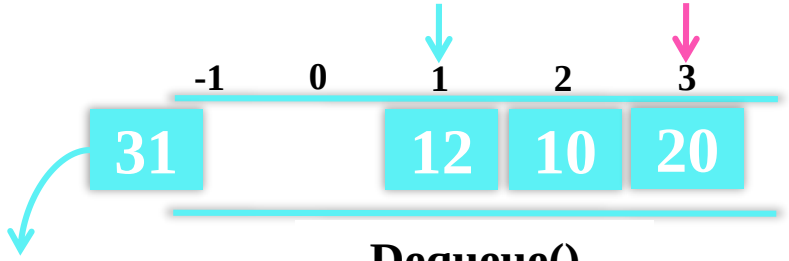
Overflow



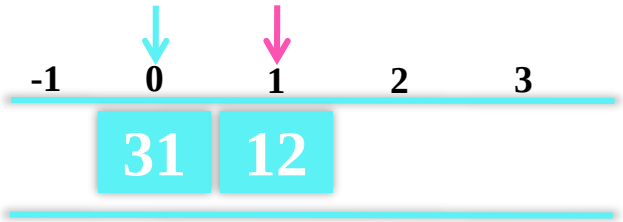
Enqueue(31)



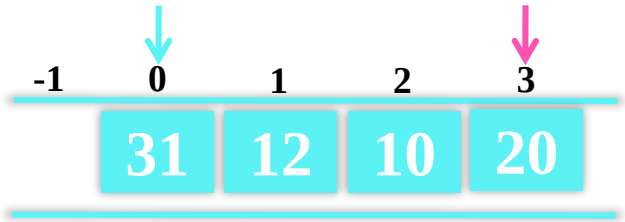
Enqueue(10)



Dequeue()

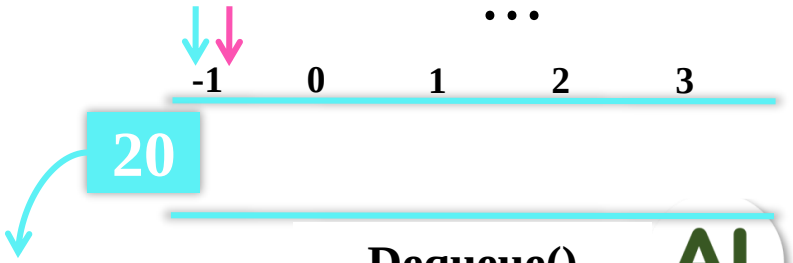


Enqueue(12)



Enqueue(20)

...



Dequeue()

Thực hiện xây dựng class Queue với các chức năng (method) sau đây

- **initialization method** nhận một input "capacity": dùng để khởi tạo queue với capacity là số lượng element mà queue có thể chứa
- .**isEmpty**(): kiểm tra queue có đang rỗng
- .**isFull**(): kiểm tra queue đã full chưa
- .**dequeue**(): loại bỏ first element và trả về giá trị đó
- .**enqueue**(value) add thêm value vào trong queue
- .**front**() lấy giá trị first element hiện tại của queue, nhưng không loại bỏ giá trị đó
- Không cần thiết phải thực hiện với các pointers như trong hình minh họa

```
queue1 = MyQueue(capacity=5)

queue1.enqueue(1)

queue1.enqueue(2)

print(queue1.isFull())
>> False

print(queue1.front())
>> 1

print(queue1.dequeue())
>> 1

print(queue1.front())
>> 2

print(queue1.dequeue())
>> 2

print(queue1.isEmpty())
>> True
```

Queue
- capacity: int - queue: list()
+ __init__(capacity) + isEmpty(): bool + isFull(): bool + dequeue(): any + enqueue(value): void + front(): any

Thực hiện xây dựng class Queue với các chức năng (method) sau đây

- **initialization method** nhận một input "capacity": dùng để khởi tạo queue với capacity là số lượng element mà queue có thể chứa
- .**isEmpty**(): kiểm tra queue có đang rỗng
- .**isFull**(): kiểm tra queue đã full chưa
- .**dequeue**(): loại bỏ first element và trả về giá trị đó
- .**enqueue**(value) add thêm value vào trong queue
- .**front**() lấy giá trị first element hiện tại của queue, nhưng không loại bỏ giá trị đó
- Không cần thiết phải thực hiện với các pointers như trong hình minh họa

Queue

- capacity: int
- queue: list()

+ __init__(capacity)
+ isEmpty(): bool
+ isFull(): bool
+ dequeue(): any
+ enqueue(value): void
+ front(): any

```
CLASS Queue
    INIT FUNCTION __init__(capacity)
        PRIVATE __capacity = capacity
        PRIVATE __queue = list()
    ENDFUNCTION

    PUBLIC FUNCTION isEmpty()
        RETURN len(__queue ) == 0
    ENDFUNCTION

    PUBLIC FUNCTION isFull()
        RETURN len(__queue ) == __capacity
    ENDFUNCTION

    PUBLIC FUNCTION dequeue()
        IF isEmpty()
            RAISE EXCEPTION("Underflow")
        ENDIF
        RETURN __queue.pop(0)
    ENDFUNCTION
...
```

```
    PUBLIC FUNCTION enqueue(value)
        IF isFull()
            RAISE EXCEPTION("Overflow")
        ENDIF
        __queue.append(value)
    ENDFUNCTION

    PUBLIC front()
        IF isEmpty()
            PRINT("Queue is empty")
            RETURN
        ENDIF
        RETURN __stack.[0]
    ENDFUNCTION

ENDCLASS
```