AI VIETNAM
All-in-One Course

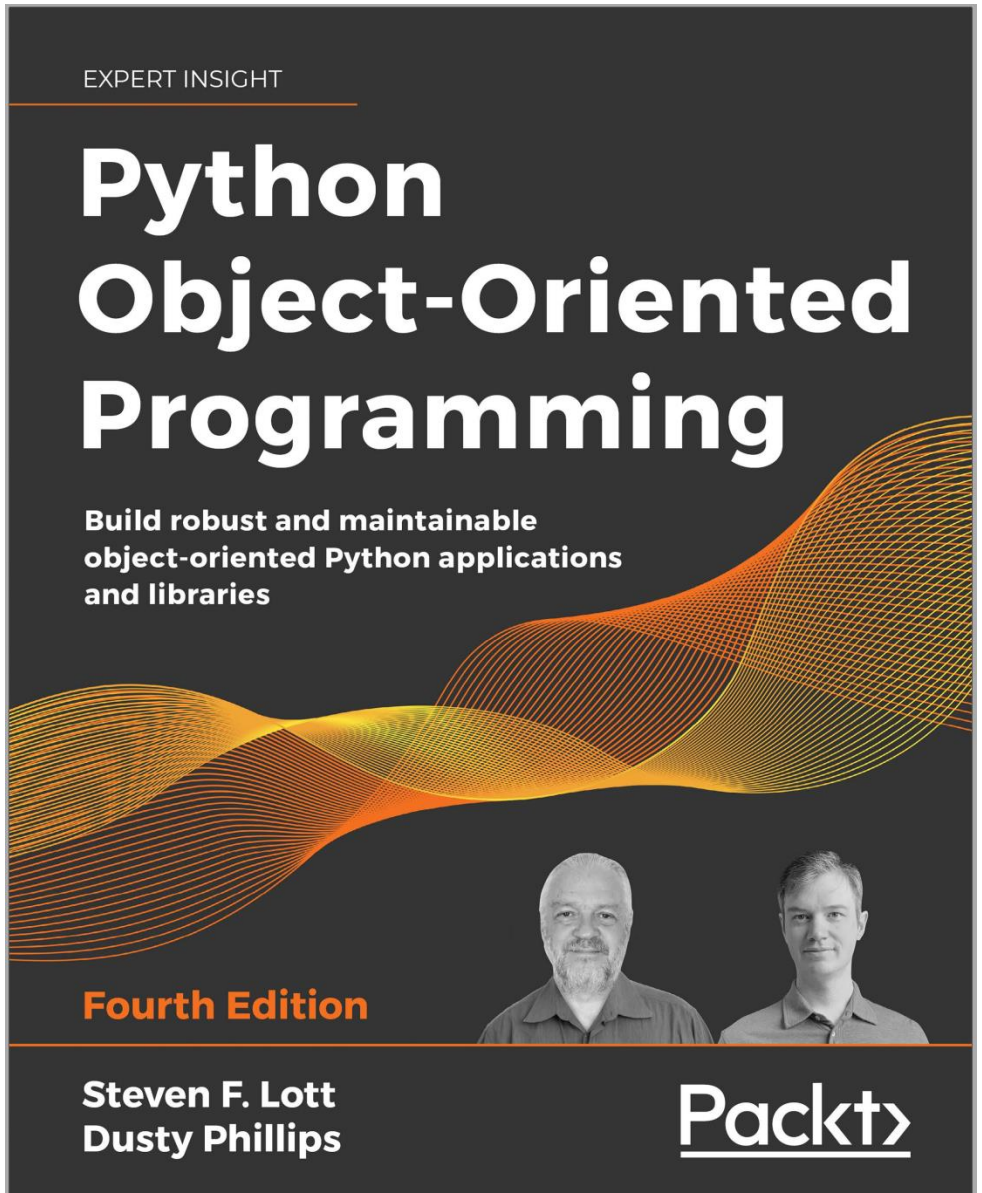
Object-Oriented Programming

(Objects and Classes)

Quang-Vinh Dinh
PhD in Computer Science

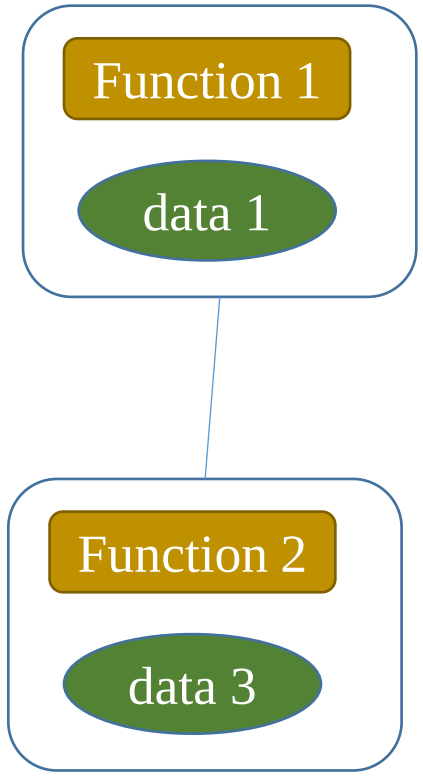
Year 2024

Reference Books

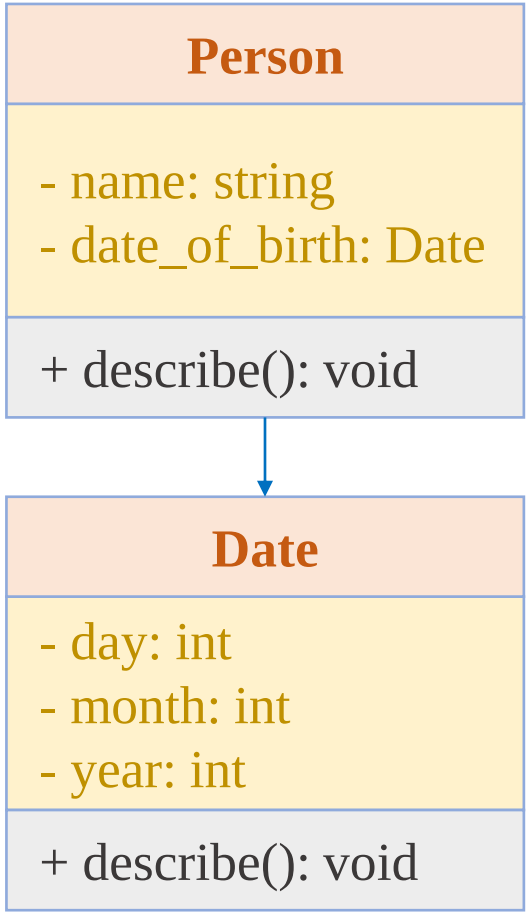


Objectives

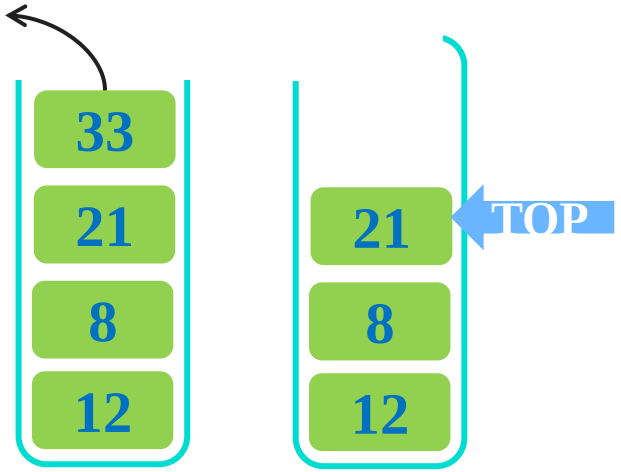
Class (Encapsulation)

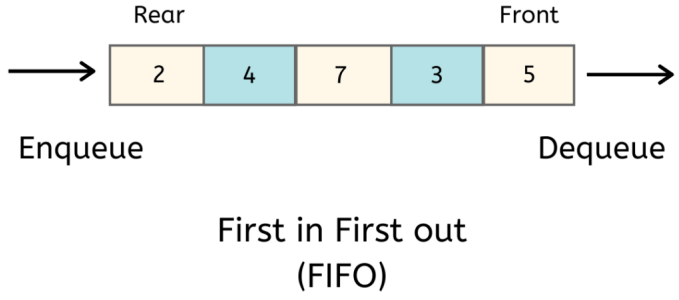


Delegation



Stack & Queue





First in First out
(FIFO)

Outline

SECTION 1

Introduction to OOP

SECTION 2

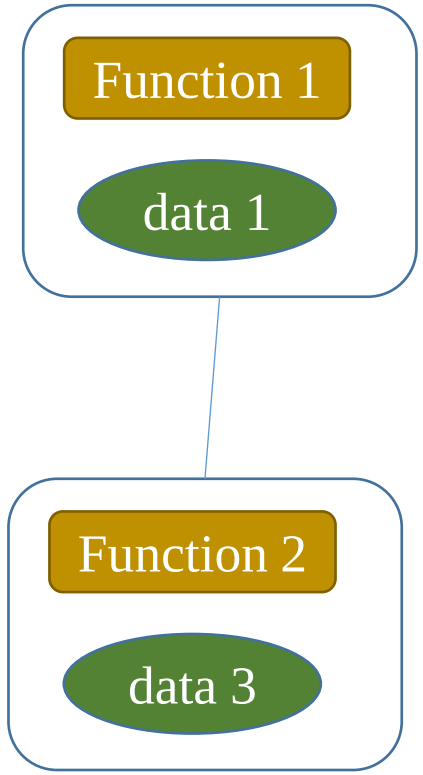
Objects and Classes

SECTION 3

Delegation

SECTION 4

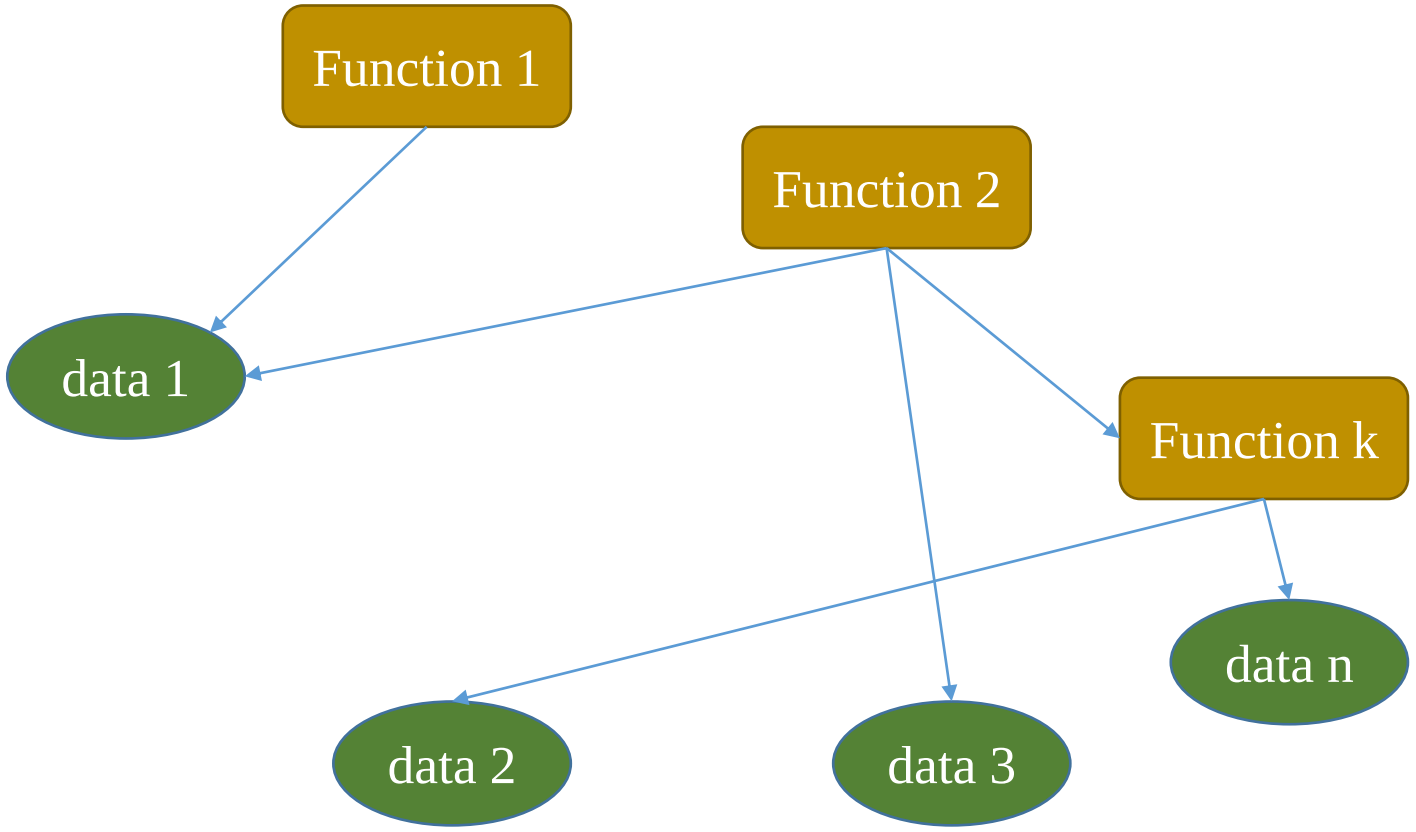
Stack and Queue

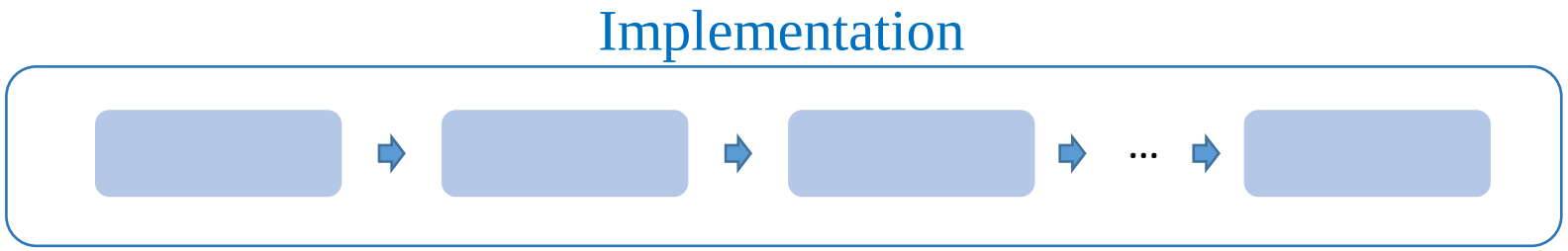


OOP Introduction

❖ Procedural programming

- Top-down approach
- Security
- Maintenance
- Reusability
- Implementation





OOP Introduction

❖ OOP programming

- Bottom-up approach
- Security
- Maintenance
- Reusability
- Implementation

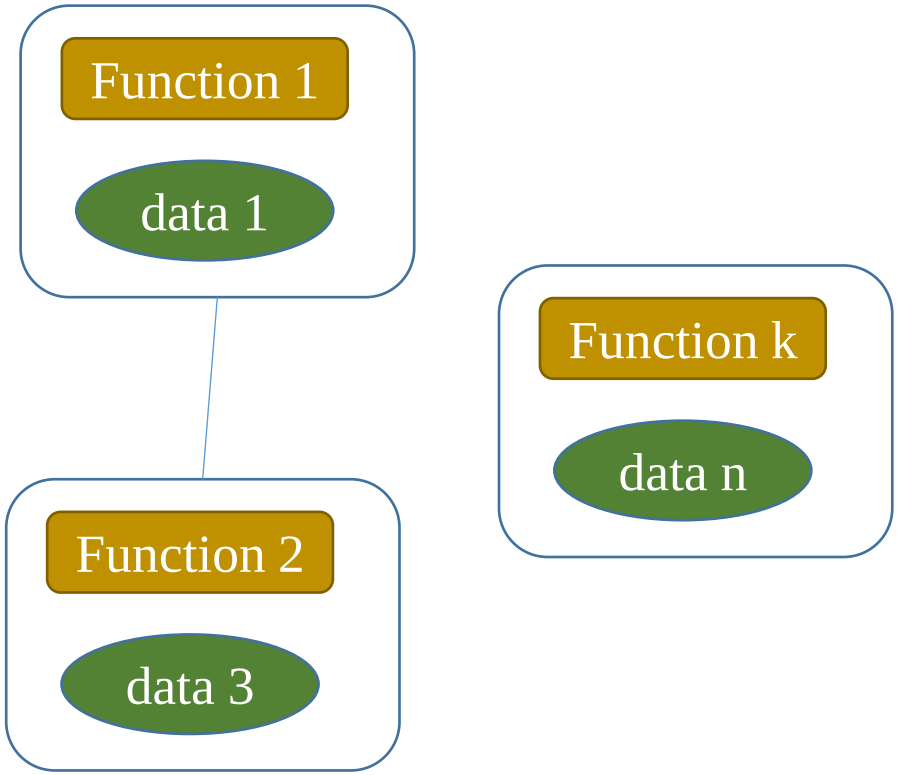
Access modifiers

Inheritance

Objects/Classes

Encapsulation

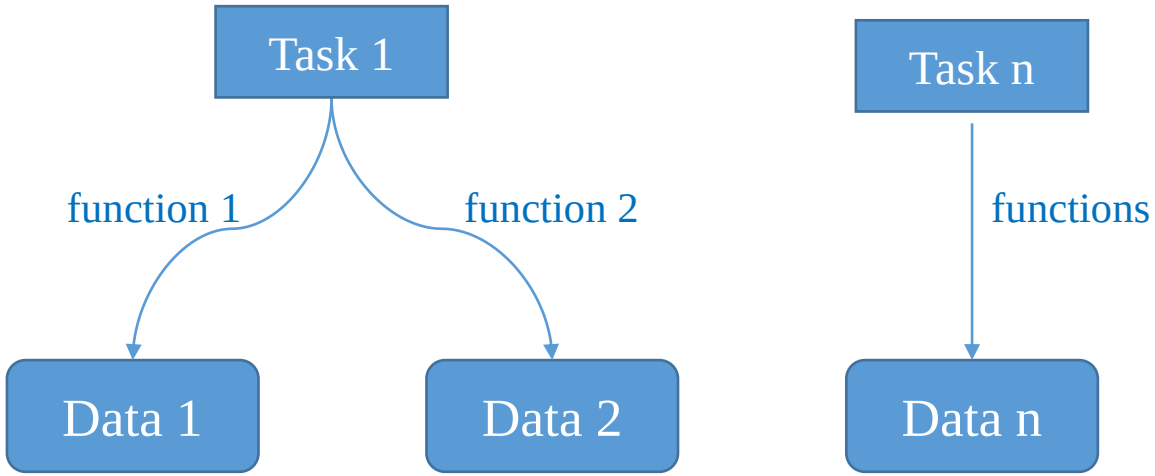
Polymorphism



Introduction

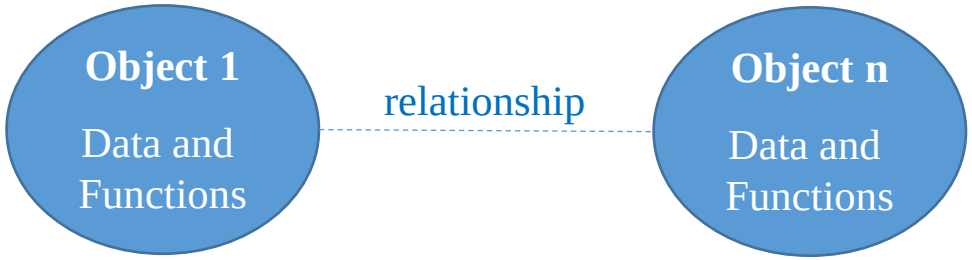
❖ What is OOP?

Writing functions that perform operations on the data



Procedural programming

Creating objects that contain both data and functions



Object-oriented programming

Introduction


```
# code segment 1
alpha = 0.01
def function1(value):
    return max(value, value*alpha)

# test
print(function1(5))
print(function1(-3))

# code segment 2
def function2(name):
    return 'Hi ' + name

# test
data3 = 'John'
print(function2(data3))
```

```
5
-0.03
Hi John
```

```
# code segment 1
alpha = 0.01
def function1(value):
    return max(value, value*alpha)

# test
print(function1(5))
print(function1(-3))

# code segment 2
def function2(name):
    return 'Hi ' + name + str(alpha)

# test
data3 = 'John'
print(function2(data3))
```

```
5
-0.03
Hi John0.01
```

❖ Example

```
class LeakyReLU:
    def __init__(self, alpha):
        self.alpha = alpha

    def __call__(self, value):
        return max(value,
                   value*self.alpha)

class User:
    def __init__(self, name):
        self.name = name

    def greet(self):
        return 'Hi ' + self.name
```

Introduction

❖ Classes and Objects

A class is a template for objects, and an object is an instance of a class.

Fruit

Strawberry
Apple
Banana

Introduction

❖ Classes and Objects

A class is a template for objects, and an object is an instance of a class.

Animal

Cat
Deer
Tiger

Introduction

❖ Classes and Objects

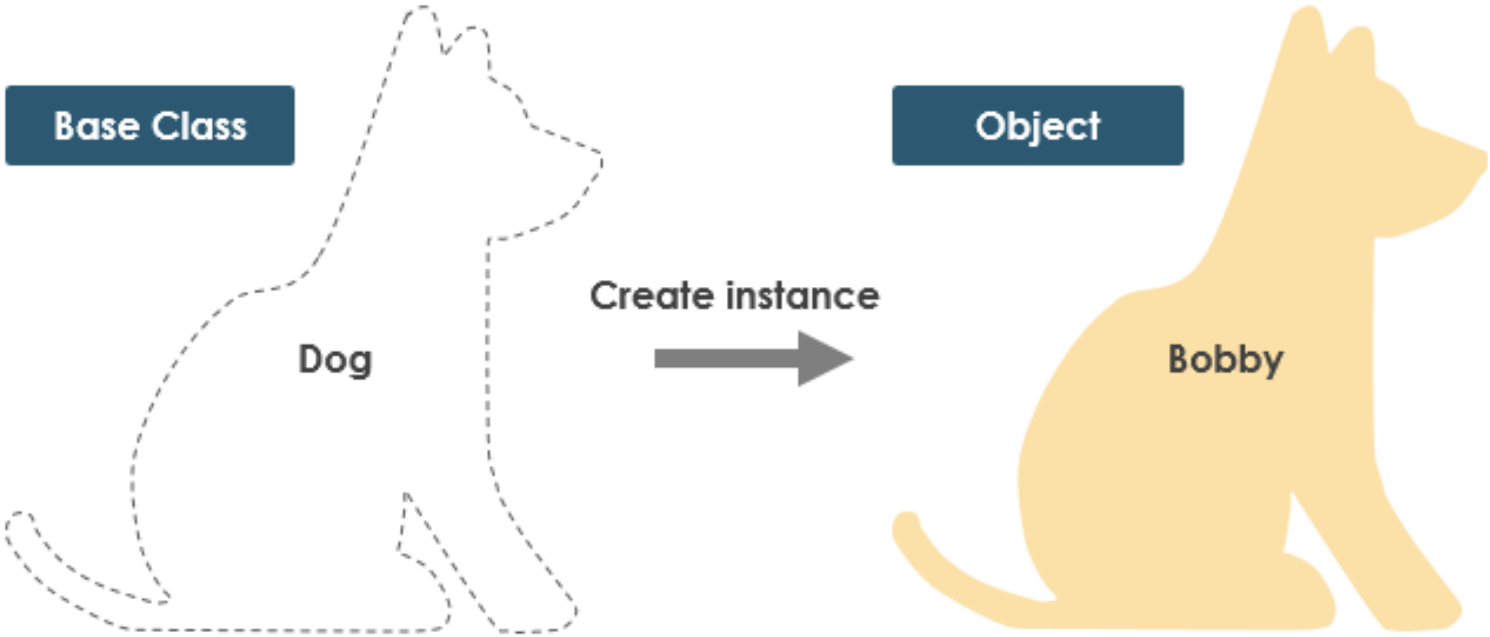
A class is a template for objects, and an object is an instance of a class.

Cat

Japanese Bobtail
Scottish Fold
Calico

Introduction

❖ Classes and Objects



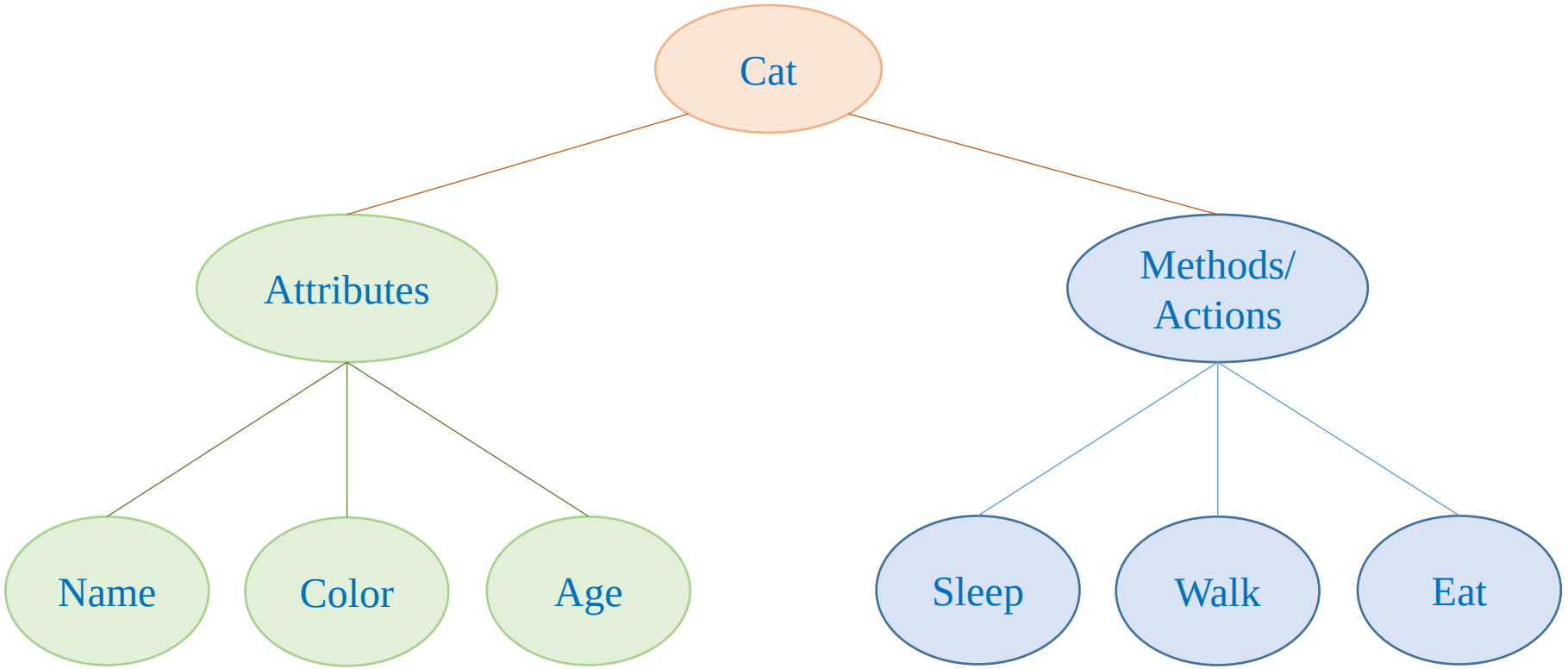
Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

Classes and Objects

❖ Abstract view



Class Diagram

Class Diagram

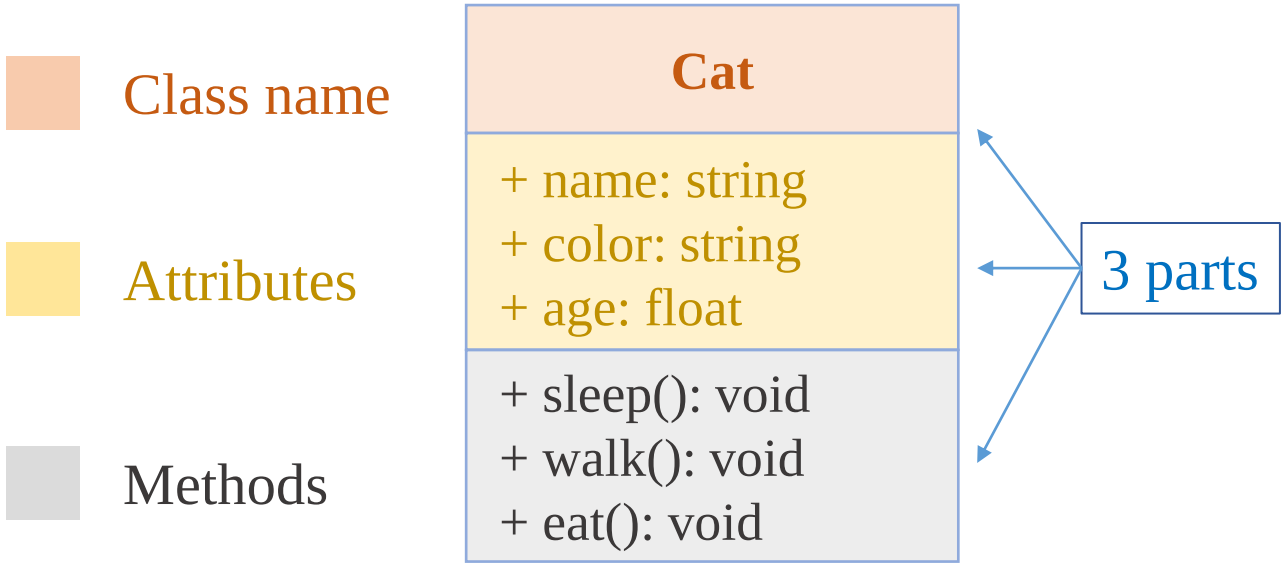
❖ Describe the structure of a system

Classes
their attributes
operations (methods)
relationships among objects

Access modifiers
- private
+ public

A cat includes a name, a color, and an age. The daily activities of the cat consists of sleeping, walking, and eating.

Draw a class diagram for the above description. All the attributes and methods are publicly accessed.



Outline

SECTION 1

Introduction to OOP

SECTION 2

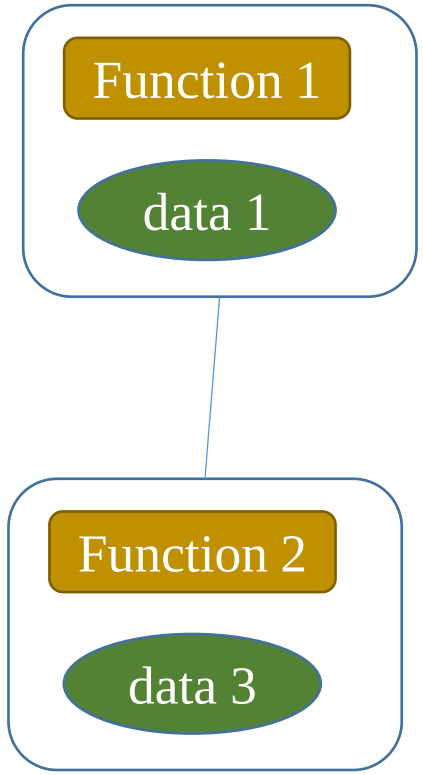
Objects and Classes

SECTION 3

Delegation

SECTION 4

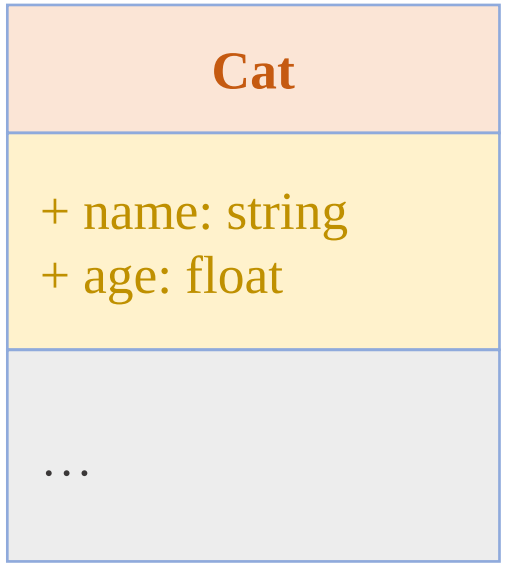
Stack and Queue



Classes and Objects

❖ Implementation using Python

A cat has a name and an age.
By default, a cat is named 'Calico' and is 0.8 years old.



class name

keyword

attribute

```
# create a class
class Cat:
    name = 'Calico'
    age = 0.8

# test: create an object
cat = Cat()
print(cat.name)
print(cat.age)
```

```
Calico
0.8
```

```
type(cat)
```

```
__main__.Cat
```

We have a new data type

```
cat = Cat()
```

variable — create an object

To access an attribute

```
cat.name
```

variable — attribute name

Classes and Objects

A cat has a name and an age.
By default, a cat is named 'Calico' and is 0.8 years old.

A class can be considered as a new space.
We can define functions in a class.

Functions insides classes are call **methods**

❖ Implementation using Python

```
# create a class
class Cat:
    name = 'unknown'
    age = -1

# test: create an object
cat = Cat()
print(cat.name)
print(cat.age)
```

```
unknown
-1
```

Cat
+ name: string + age: float
...

```
# create a class
class Cat:
    name = 'unknown'
    age = -1

    def set_init_values(input_name, input_age):
        name = input_name
        age = input_age

# test: create an object
cat = Cat()
cat.set_init_values('Calico', 0.8)
print(cat.name)
print(cat.age)
```

```
---------------------------------------------------------
TypeError                                 Traceback
```

Not that easy!
Let's find out!

Classes and Objects

❖ A step inside classes in Python

```
# create a class
class Cat:
    name = 'unknown'
    age = -1

# test: create an object
cat = Cat()
print(cat.name)
print(cat.age)
```

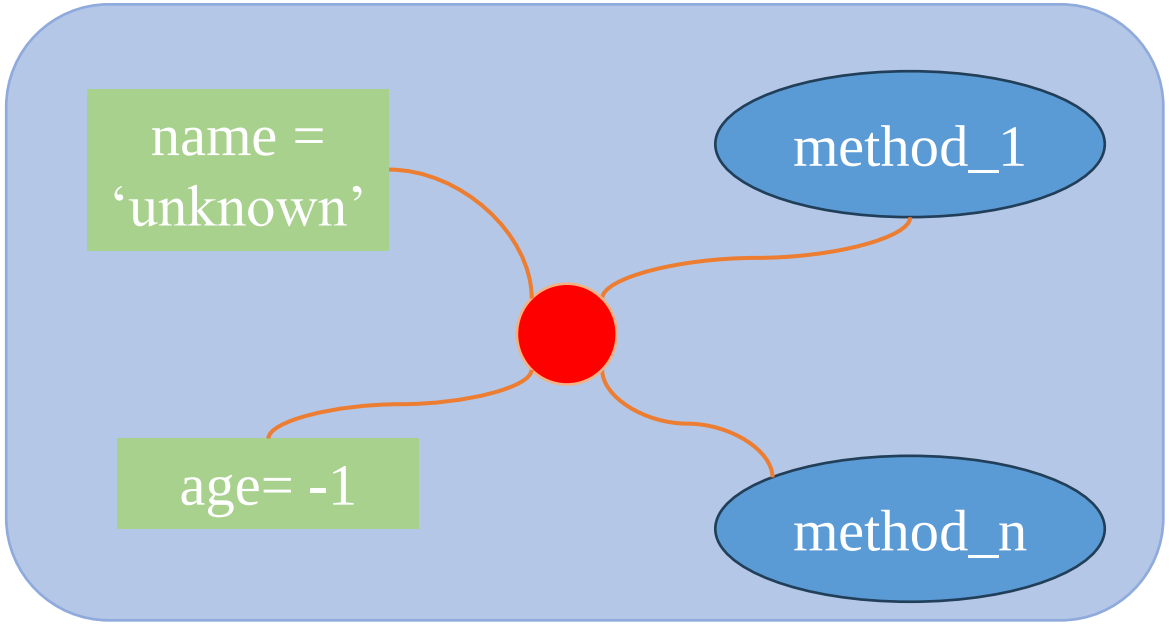
```
unknown
-1
```

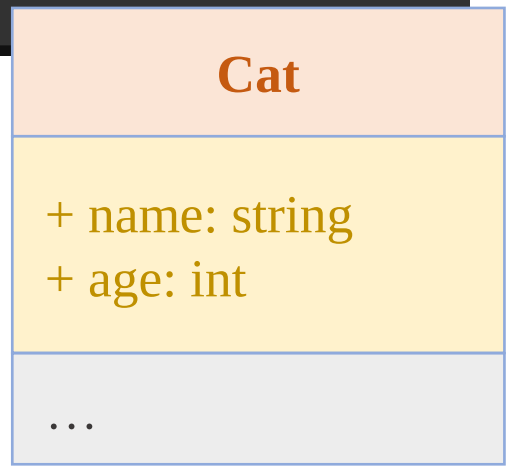
class

attributes

methods

self





To access attributes and methods inside a class

Using `self.name`
`self.age`
`self.method_1(…)`
`self.method_n(…)`

`self` is always the first parameter of a method

Classes and Objects

A cat has a name and an age.
By default, a cat is named 'Calico' and is 0.8 years old.

❖ Back to our problem

Cat
+ name: string + age: float
…

Using self.name
self.age
self.method_1(…)
self.method_n(…)

self is always the first parameter of a method

How to improve more?

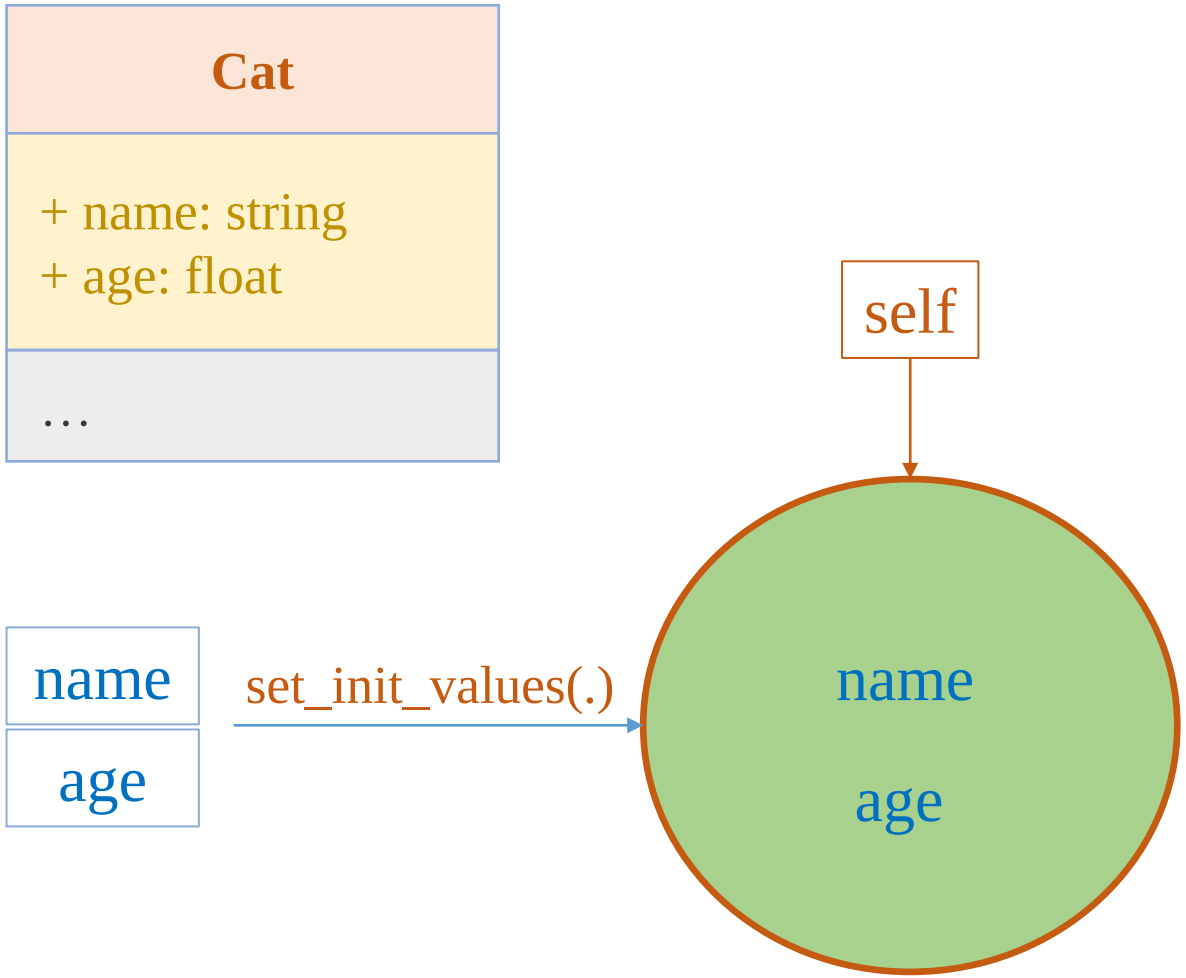
```
# create a class
class Cat:
    name = 'unknown'
    age = -1

    def set_init_values(self, input_name, input_age):
        self.name = input_name
        self.age = input_age

# test: create an object
cat = Cat()
cat.set_init_values('Calico', 0.8)
print(cat.name)
print(cat.age)
```

```
Calico
0.8
```

Classes and Objects

A cat has a name and an age.
By default, a cat is named 'Calico' and is 0.8 years old.

❖ Naming efficiently



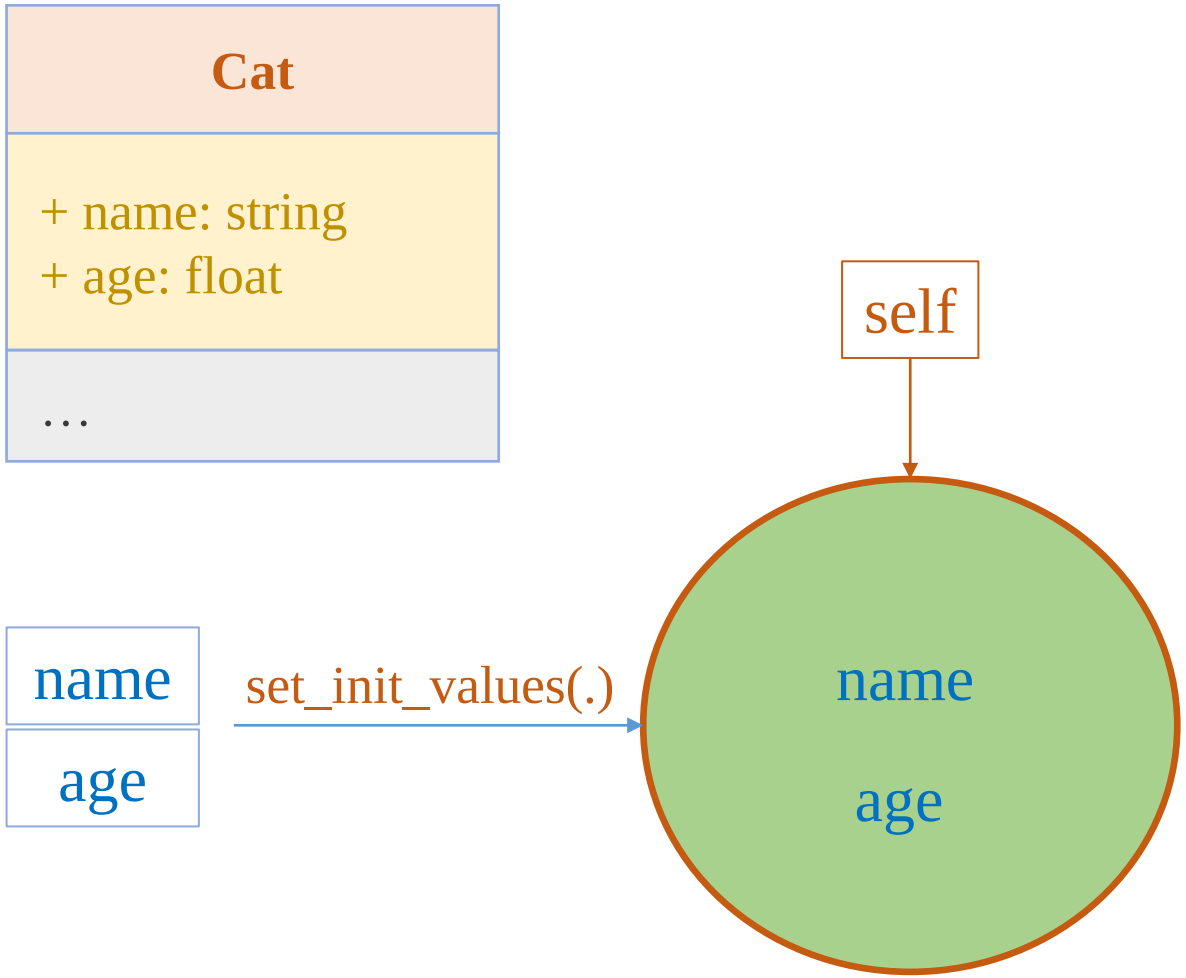

```python
# create a class
class Cat:
    name = 'unknown'
    age = -1

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

# test: create an object
cat = Cat()
cat.set_init_values('Calico', 0.8)
print(cat.name)
print(cat.age)
```

```
Calico
0.8
```

Classes and Objects

A cat has a name and an age.
By default, a cat is named 'Calico' and is 0.8 years old.

❖ **We can create attributes with self**




```python
# create a class
class Cat:
    def set_init_values(self, name, age):
        self.name = name
        self.age = age

# test: create an object
cat = Cat()
cat.set_init_values('Calico', 0.8)
print(cat.name)
print(cat.age)
```

```
Calico
0.8
```

Classes and Objects

❖ Some more methods

Cat
+ name: string + age: float
+ set_init_values(name, age) + increase_age(age) + describe()

name = ‘Calico’

age= 0.8

name = ‘Calico’

age= 1.0

Observation?

```python
# create a class
class Cat:
    def set_init_values(self, name, age):
        self.name = name
        self.age = age

    def increase_age(self, age):
        self.age = self.age + age

    def describe(self):
        print(f'Name: {self.name} - Age: {self.age}')

# test: create an object
cat = Cat()
cat.set_init_values('Calico', 0.8)
cat.describe()

cat.increase_age(0.2)
cat.describe()
```

```
Name: Calico - Age: 0.8
Name: Calico - Age: 1.0
```

Make it simpler


```python
# create a class
class Cat:
    def set_init_values(self, name, age):
        self.name = name
        self.age = age

    def increase_age(self, age):
        self.age = self.age + age

    def describe(self):
        print(f'Name: {self.name} - Age: {self.age}')

# test: create an object
cat = Cat()
cat.set_init_values('Calico', 0.8)
cat.describe()

cat.increase_age(0.2)
cat.describe()
```

```
Name: Calico - Age: 0.8
Name: Calico - Age: 1.0
```

```python
# create a class
class Cat:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def increase_age(self, age):
        self.age = self.age + age

    def __call__(self):
        print(f'Name: {self.name} - Age: {self.age}')

# test: create an object
cat = Cat('Calico', 0.8)
cat()

cat.increase_age(0.2)
cat()
```

```
Name: Calico - Age: 0.8
Name: Calico - Age: 1.0
```

To sum up

❖ Up to this point

The __init__() function is called automatically every time the class is being used to create a new object.

The self parameter is a reference to the current instance of the class.

__call__() function: instances behave like functions and can be called like a functions.

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

    def sum(self):
        return self.x + self.y

    def __call__(self):
        return self.x*self.y
```

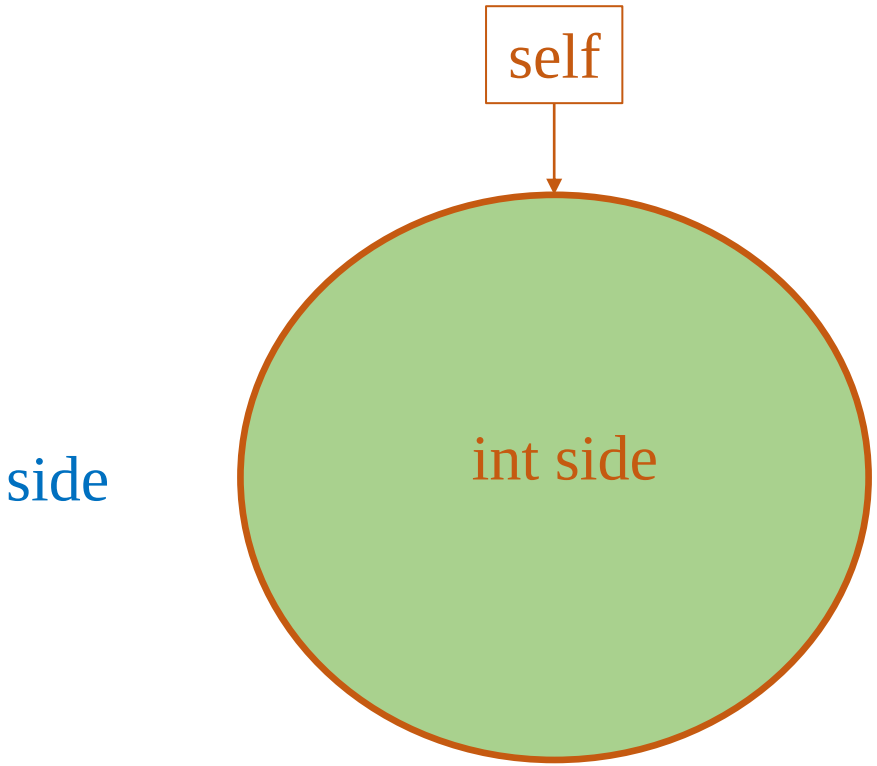
```python
point = Point(4, 5)
print(point.sum())
print(point())
```

```
9
20
```

self Keyword

Must be the first argument of methods

Used to create and access data members




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

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

# test sample: side=5 -> 25
square = Square(5)
area = square.compute_area()
print(f'Square area is {area}')
```

```
Square area is 25
```

Classes and Objects

Cat
+ name: string + color: string + age: float
...

```python
class Cat:
    def __init__(self, name, color, age):
        self.name = name
        self.color = color
        self.age = age

# test
cat = Cat('Calico', 'Black, white, and brown', 2)
print(cat.name)
print(cat.color)
print(cat.age)
```

```
Calico
Black, white, and brown
2
```

We have a new data type

cat = Cat('Calico', 'BW', 2)

variable — create an object

Naming conventions

For class names

Including words concatenated

Each word starts with upper case

For attribute names

Using nouns

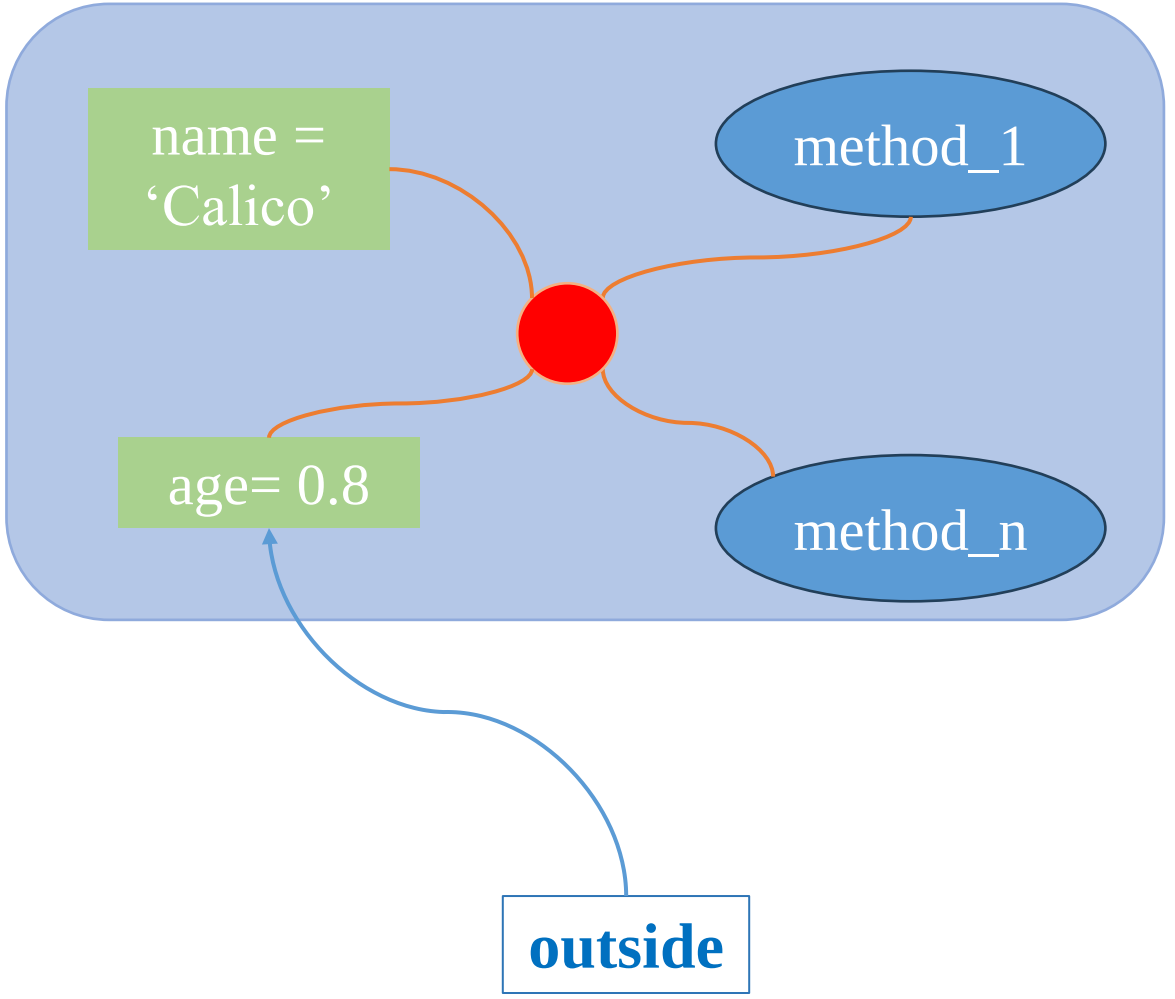
Words are connected using underscores

To continue

Any problem?

```python
# create a class
class Cat:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __call__(self):
        print(f'Name: {self.name} - Age: {self.age}')

# test: create an object
cat = Cat('Calico', 0.8)
cat()

cat.age = cat.age + 0.3
cat()
```

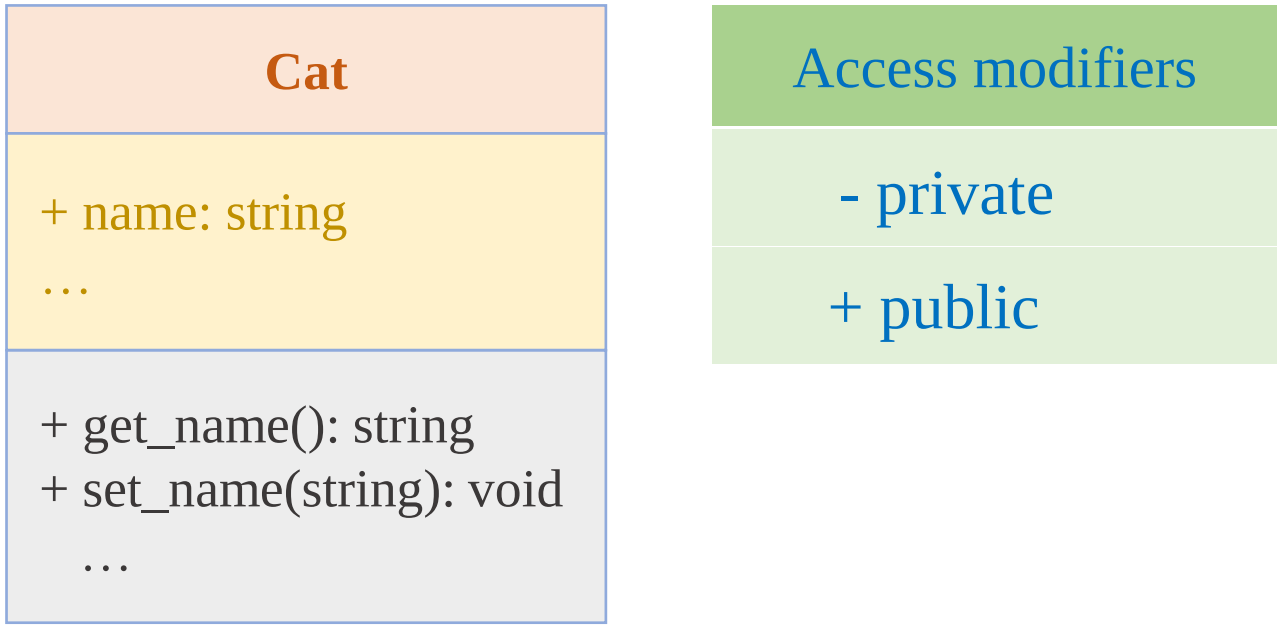
```
Name: Calico - Age: 0.8
Name: Calico - Age: 1.1
```



Classes and Objects

Problem and Solution:

Step 1 – implement getter and setter methods




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

    def get_name(self):
        return self.name

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

# test
cat = Cat('Calico')
print(cat.get_name())

cat.set_name('Japanese Bobtail')
print(cat.get_name())
```

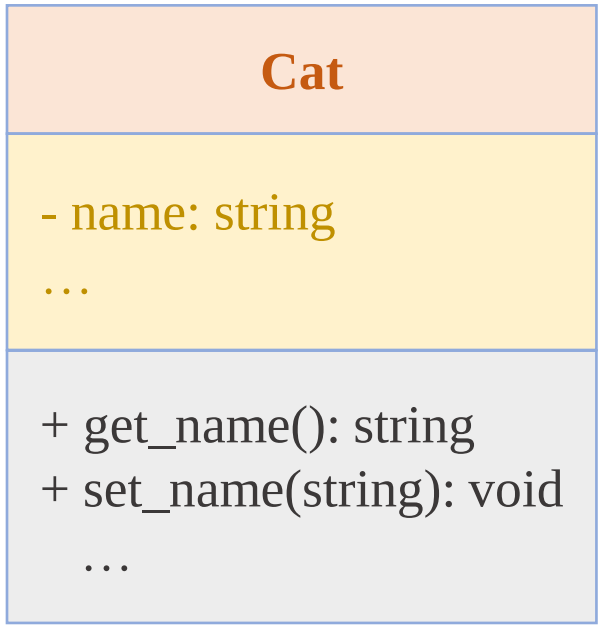
```
Calico
Japanese Bobtail
```

Classes and Objects

Solution: Step 2

Using private for attributes

Access modifiers
- private
+ public




```
print(cat.__name)
```

```
---------------------------------------------------------------
AttributeError                                Traceback (m
Cell In[4], line 1
----> 1 print(cat.__name)

AttributeError: 'Cat' object has no attribute '__name'
```

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

    def get_name(self):
        return self.__name

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

# test
cat = Cat('Calico')
print(cat.get_name())

cat.set_name('Japanese Bobtail')
print(cat.get_name())
```

```
Calico
Japanese Bobtail
```

Classes and Objects

Takeaways

Use the private access modifiers for typical attributes

Create getter and setter methods to access the attributes

Use the public access modifiers for the getter and setter functions

Access modifiers
- private
+ public

A cat includes a name, a color, and an age.

Cat
- name: string - color: string - age: float
+ get_name(): string + set_name(string): void + get_color(): string + set_color(string): void + get_age(): int + set_age(int): void

Outline

SECTION 1

Introduction to OOP

SECTION 2

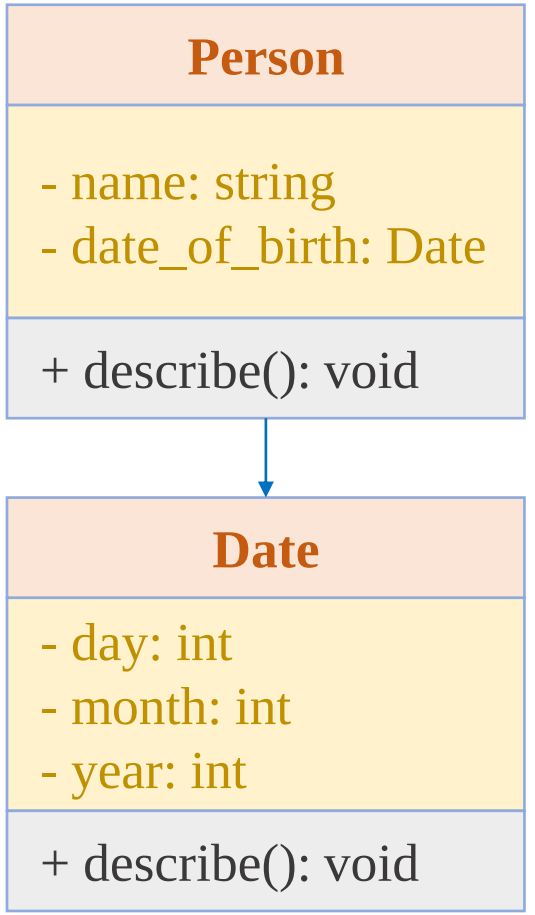
Objects and Classes

SECTION 3

Delegation

SECTION 4

Stack and Queue



Class Data Type

❖ Using a class as a data type

A person comprises a name in string and a date of birth. A date consists of day, month, and year.

Write a function to check if two people have the same name.

Write a function to check if two people have the same date of birth.

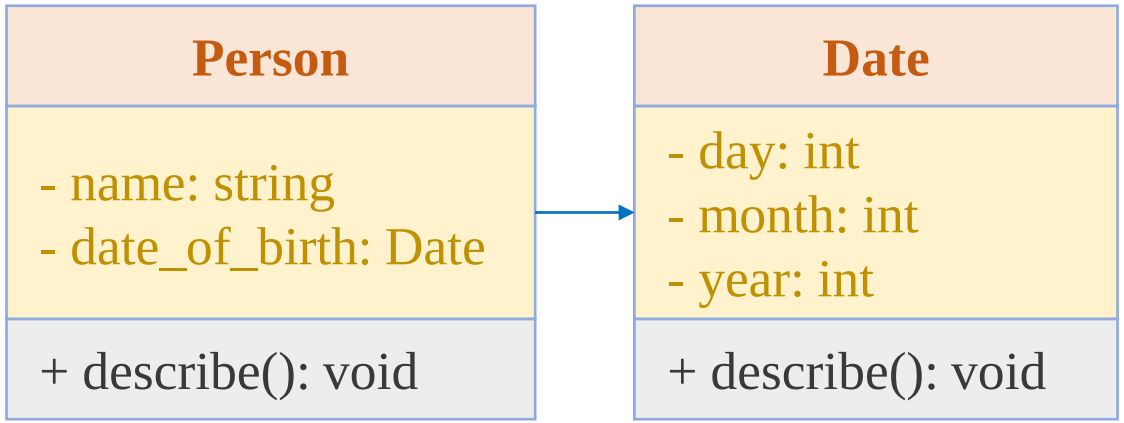
Draw a class diagram and implement in Python

Class Data Type

❖ Using a class as a data type

A person comprises a name in string and a date of birth. A date consists of day, month, and year.

Draw a class diagram and implement in Python

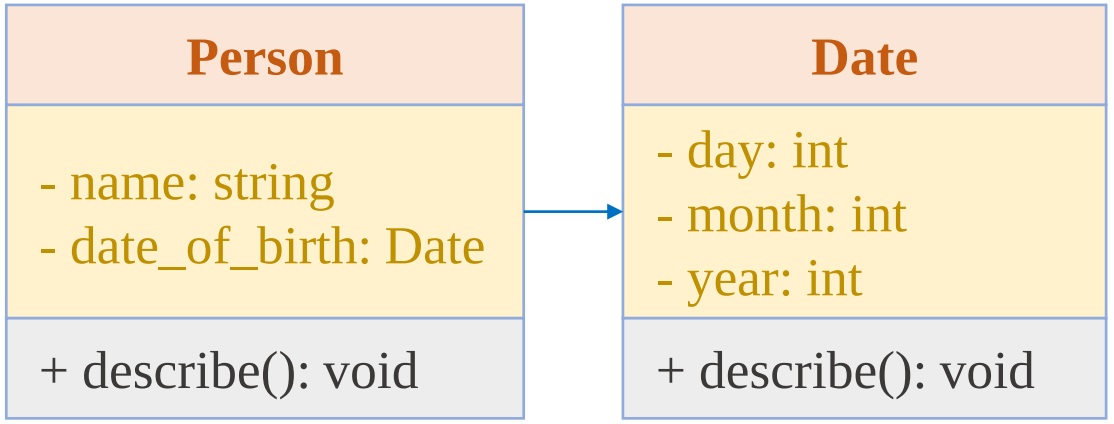



```python
class Date:
    def __init__(self, day, month, year):
        self.__day = day
        self.__month = month
        self.__year = year

    def get_day(self):
        return self.__day

    def get_month(self):
        return self.__month

    def get_year(self):
        return self.__year
```

Class Data Type

❖ Using a class as a data type

A person comprises a name in string and a date of birth. A date consists of day, month, and year.



Using Date as a data type

```python
class Person:
    def __init__(self, name, date_of_birth):
        self.__name = name
        self.__date_of_birth = date_of_birth

    def describe(self):
        # print name
        print(self.__name)

        # print date
        day = self.__date_of_birth.get_day()
        month = self.__date_of_birth.get_month()
        year = self.__date_of_birth.get_year()
        print(f'{day}/{month}/{year}')
```

```python
date = Date(10, 1, 2000)
peter = Person('Peter', date)
peter.describe()
```

```
Peter
10/1/2000
```

Class Data Type


```python
class Date:
    def __init__(self, day, month, year):
        self.__day = day
        self.__month = month
        self.__year = year

    def get_day(self):
        return self.__day

    def get_month(self):
        return self.__month

    def get_year(self):
        return self.__year

    def describe(self):
        print(f'{self.__day}/{self.__month}/{self.__year}')
```

delegation

```python
class Person:
    def __init__(self, name, date_of_birth):
        self.__name = name
        self.__date_of_birth = date_of_birth

    def describe(self):
        # print name
        print(self.__name)

        # print date
        self.__date_of_birth.describe()
```

```python
date = Date(10, 1, 2000)
peter = Person('Peter', date)
peter.describe()
```

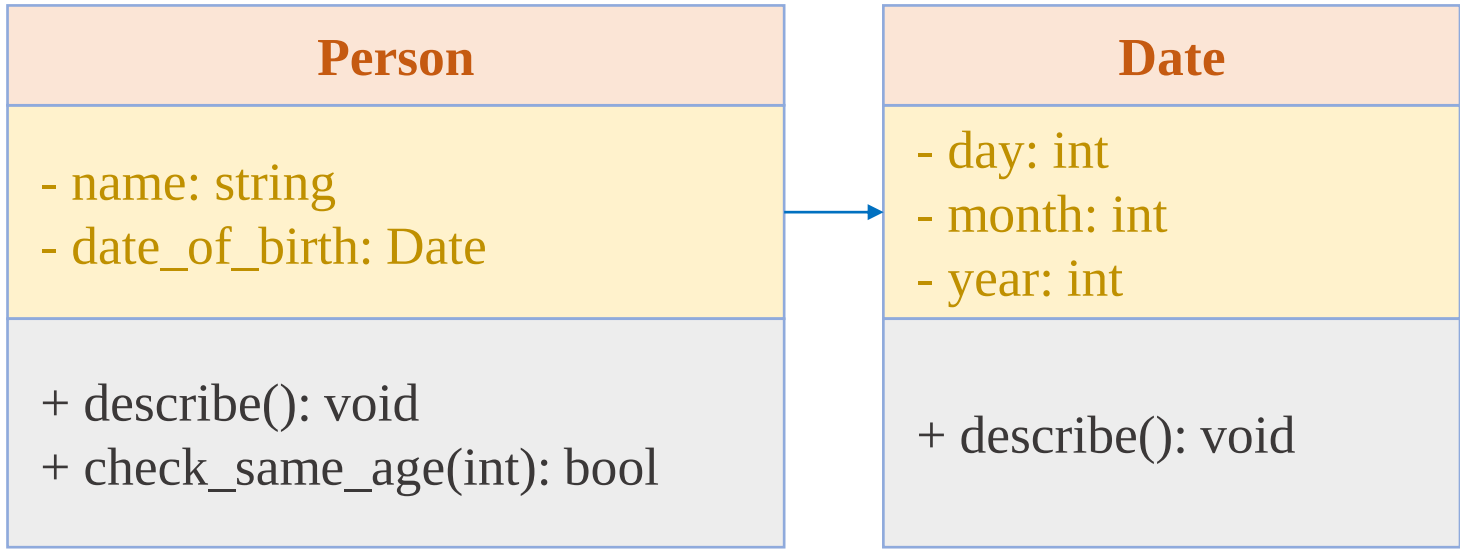
```
Peter
10/1/2000
```

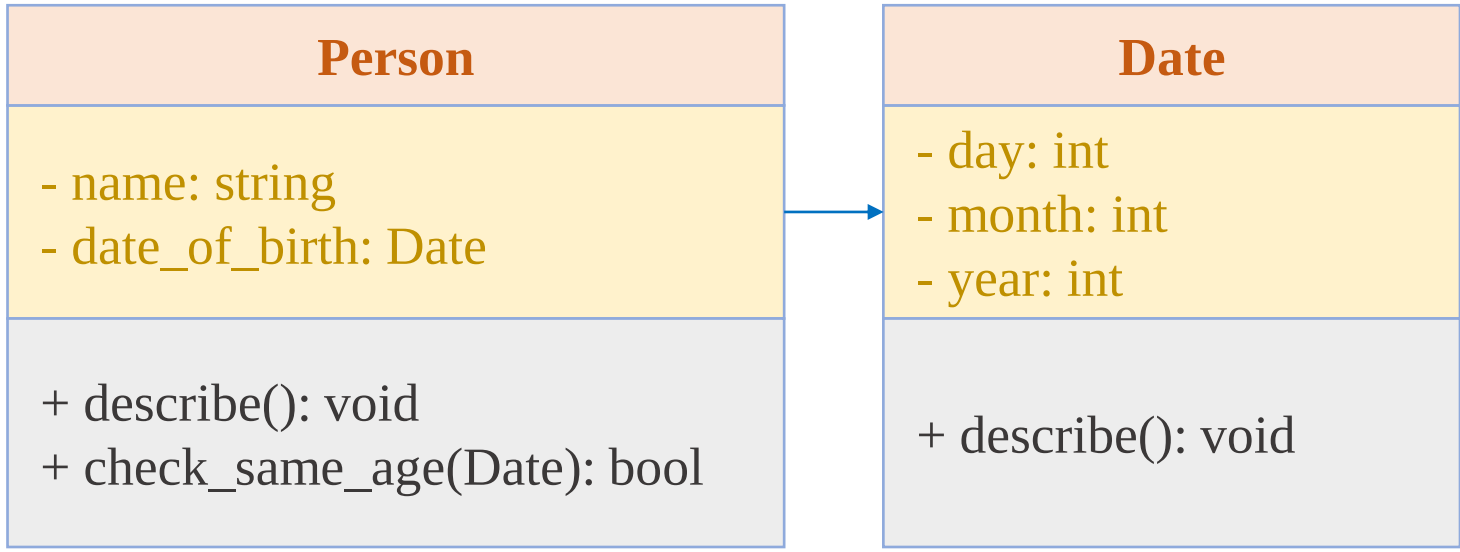
Class Data Type

❖ Using a class as a data type

A person comprises a name in string and a date of birth. A date consists of day, month, and year.

Write a function to check if two people have the same date of birth.





Outline

SECTION 1

Introduction to OOP

SECTION 2

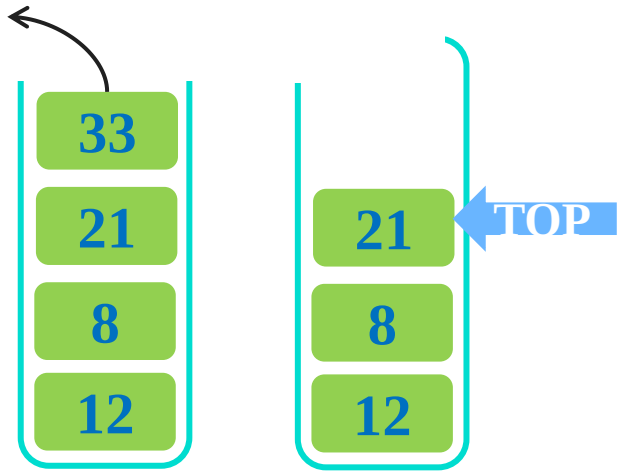
Objects and Classes

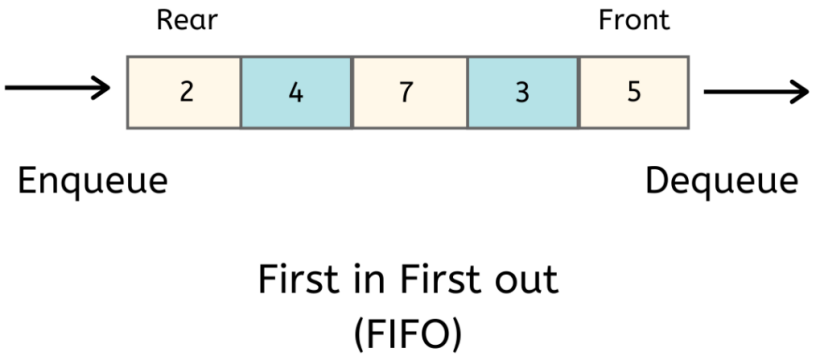
SECTION 3

Delegation

SECTION 4

Stack and Queue

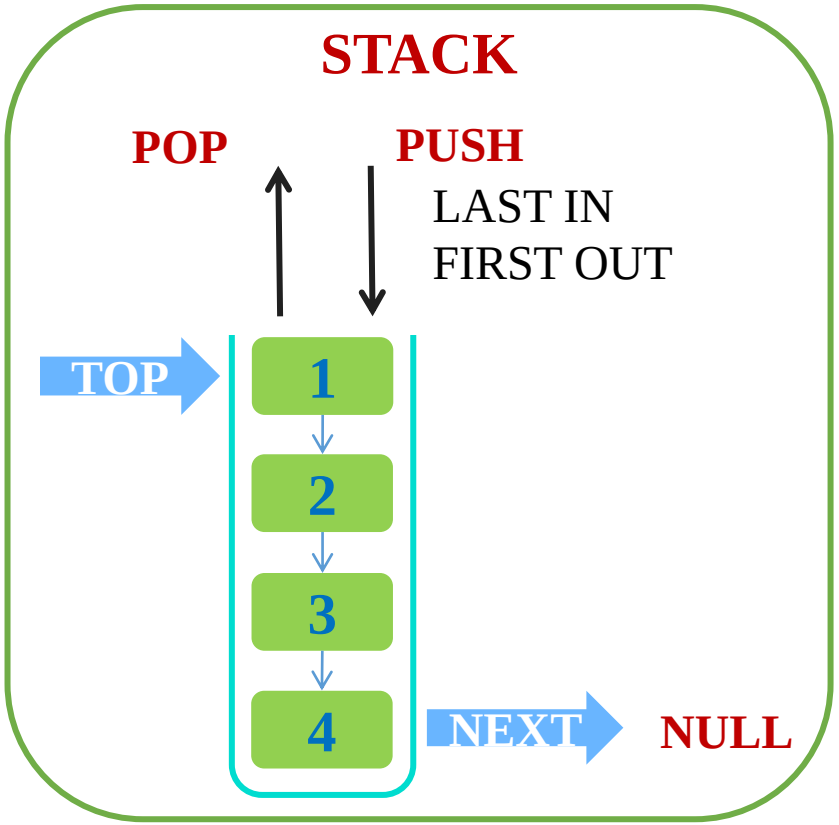


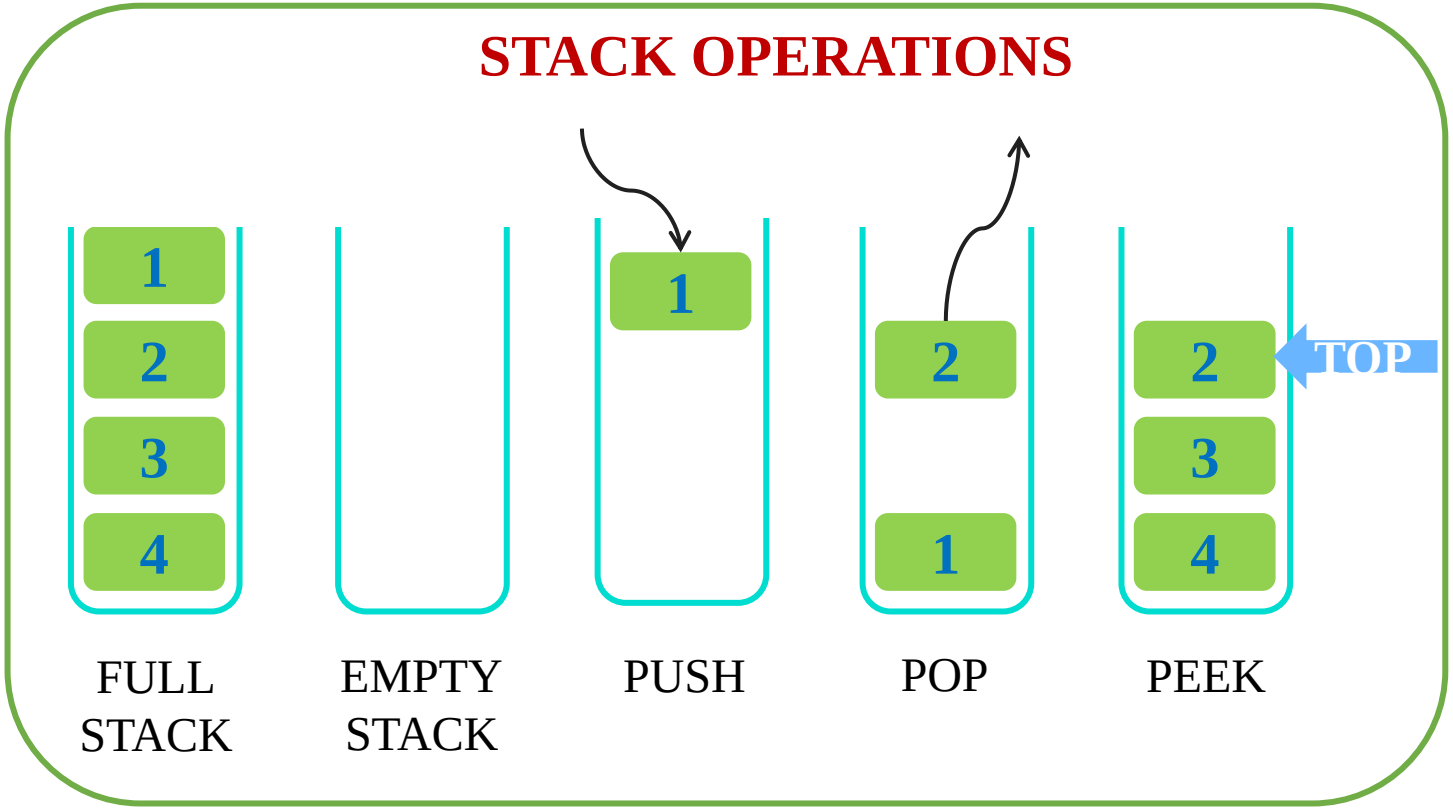


First in First out
(FIFO)

Stack Data Structure Using List

- LIFO (Last in first out)
- Element are inserted and extracted only from ONE end

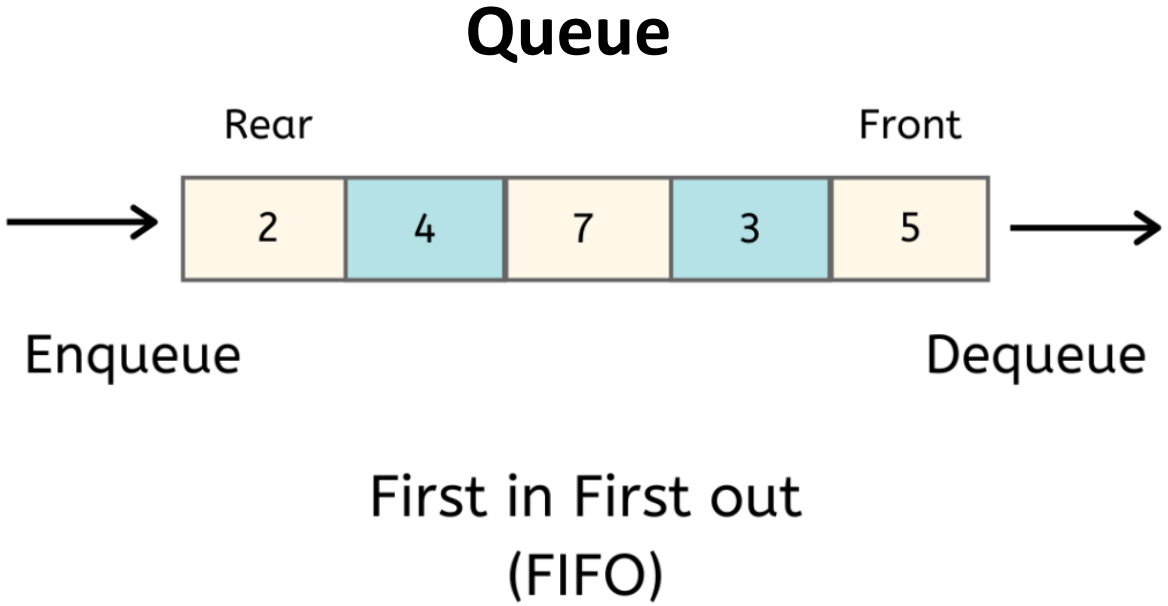


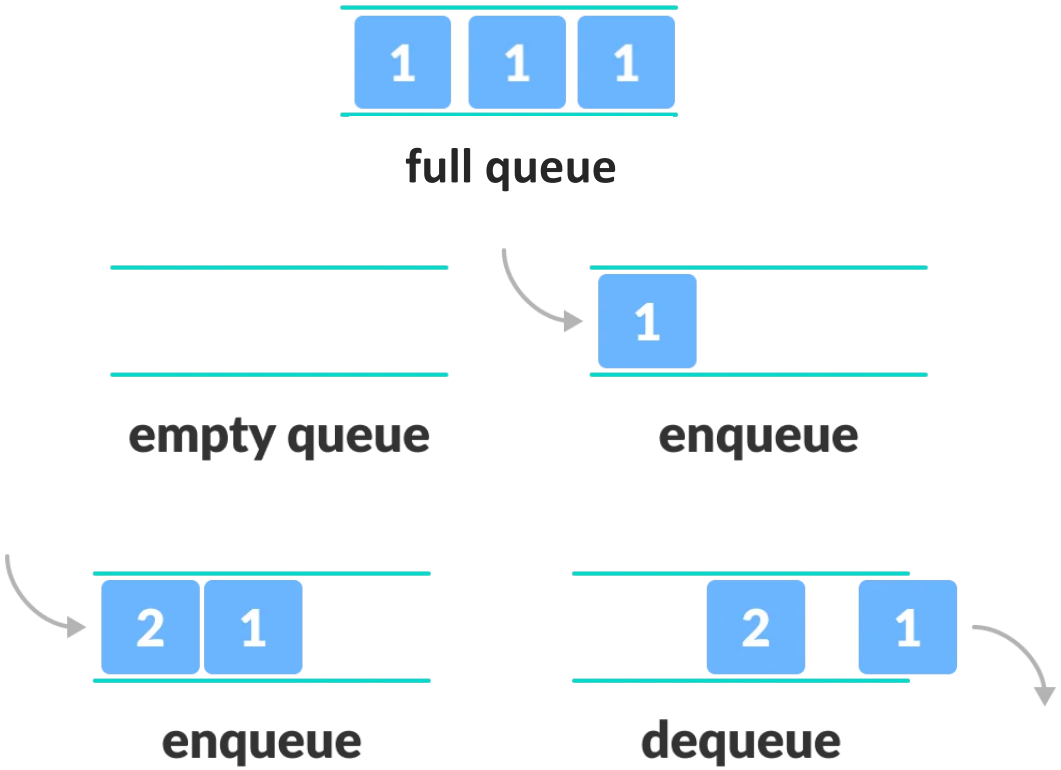


Queue Data Structure

❖ Introduction

- Operate in a **FIFO (First in First out)** context
- **Element** are **inserted** (enqueue) and **extracted** (dequeue) happens at **OPPOSITE** ends



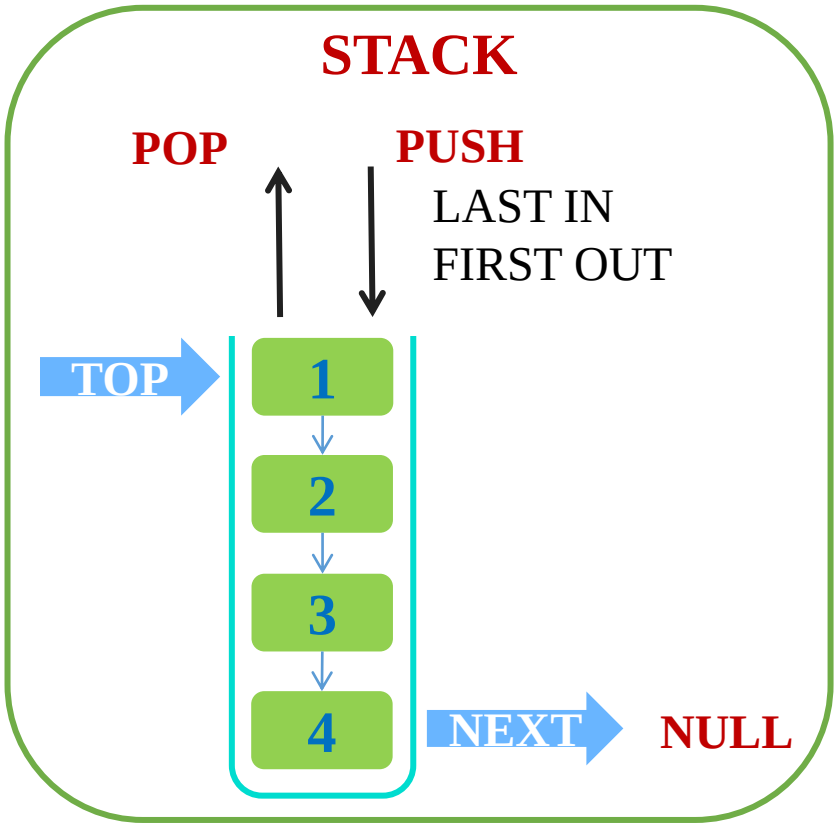


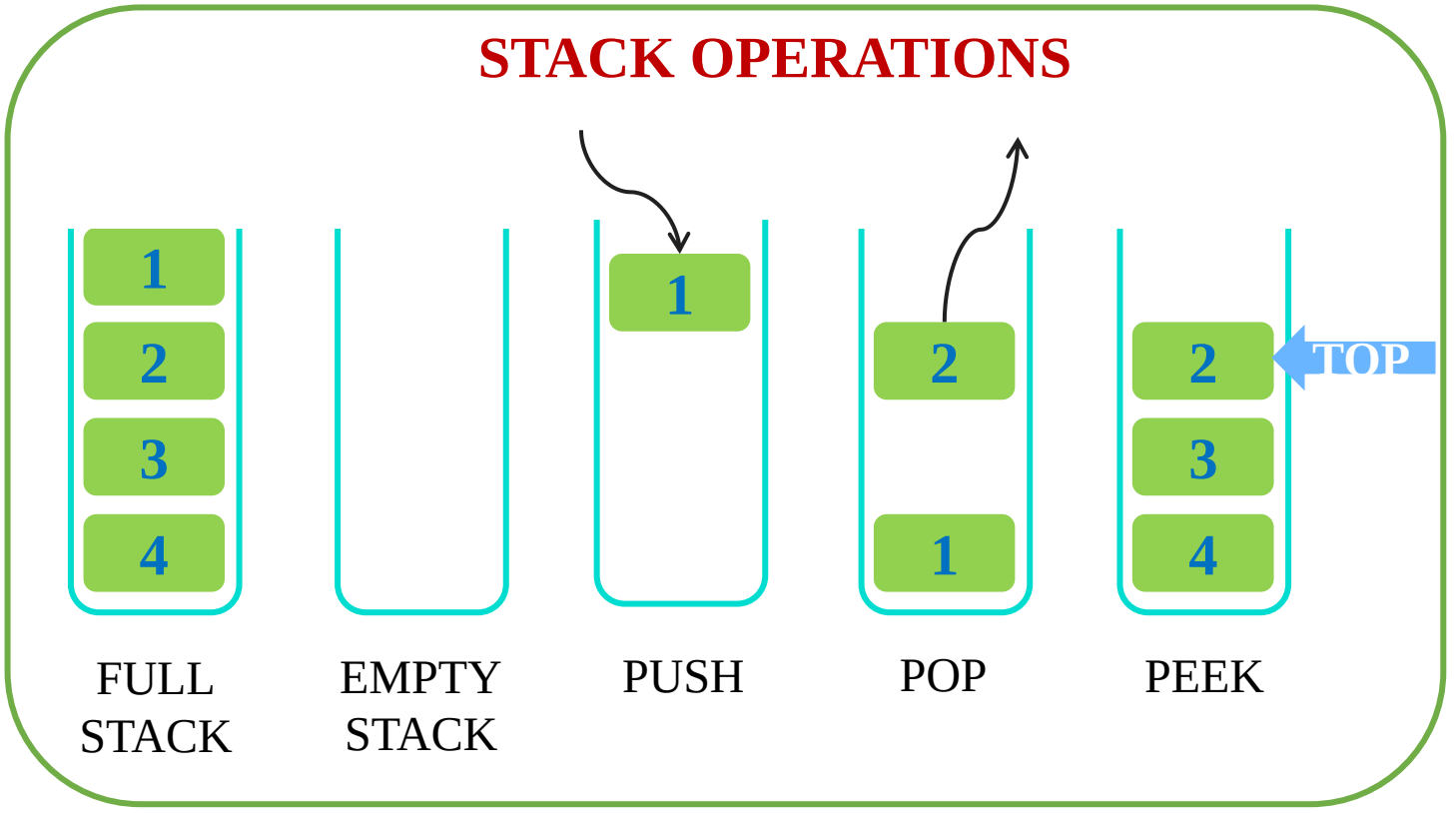

QUIZ TIME

Stack Data Structure Using List

- LIFO (Last in first out)
- Element are inserted and extracted only from ONE end

Basic Operations of Stack
- push(value): Add an element to the top of a stack
- pop(): Remove an element from the top of a stack
- is_empty(): Check if the stack is empty
- is_full(): Check if the stack is full
- peek: Get the value of the top element without removing it





List Review

❖ Add an element

data = | 6 | 5 | 7 | 1 | 9 | 2 |

data.append(4) # thêm 4 vào vị trí cuối list

data = | 6 | 5 | 7 | 1 | 9 | 2 | 4 |

```
1 data = [6, 5, 7, 1, 9, 2]
2 print(data)
3 data.append(4)
4 print(data)
```

```
[6, 5, 7, 1, 9, 2]
[6, 5, 7, 1, 9, 2, 4]
```

❖ Deleting an element

data = | 6 | 5 | 7 | 1 | 9 | 2 |

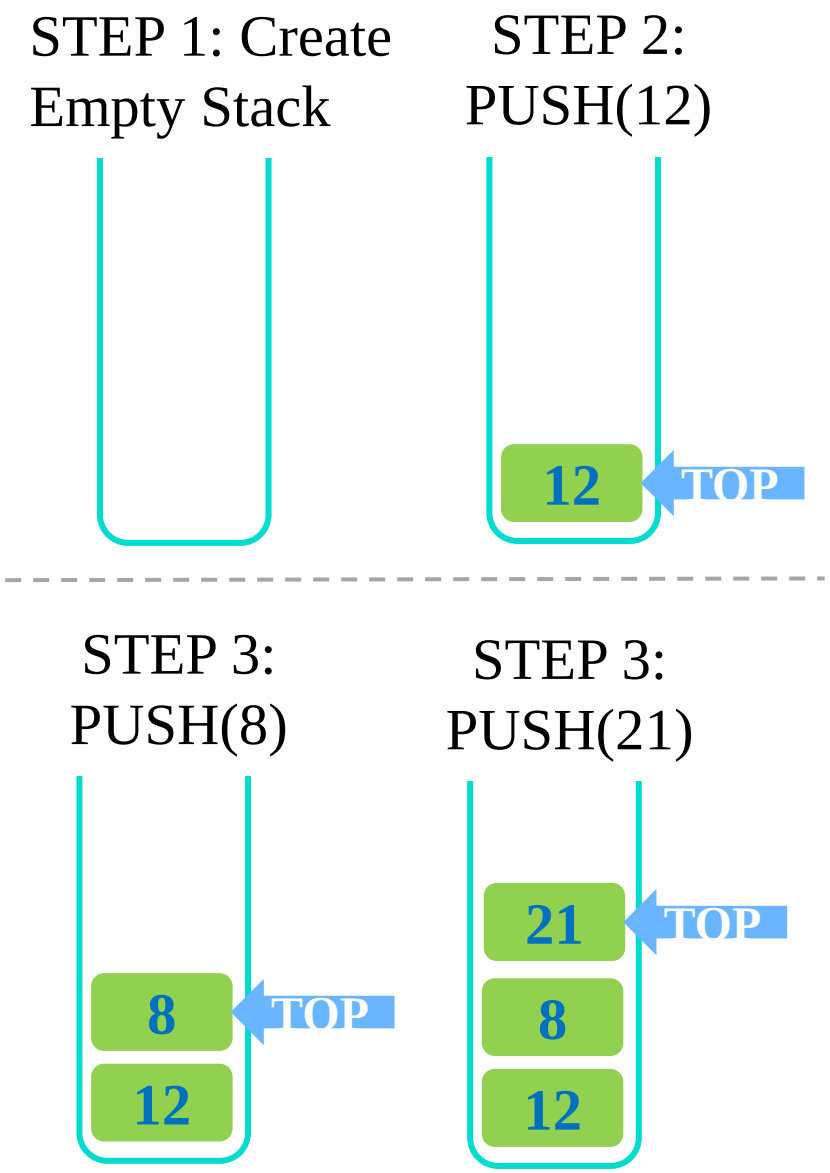
data.pop(-1) # xóa phần tử ở vị trí cuối

data = | 6 | 5 | 7 | 1 | 9 |

```
1 data = [6, 5, 7, 1, 9, 2]
2 print(data)
3 data.pop(-1)
4 print(data)
```

```
[6, 5, 7, 1, 9, 2]
[6, 5, 7, 1, 9]
```

Stack Data Structure Using List



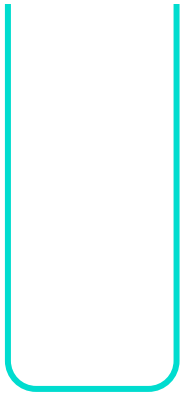

```python
class MyStack:
    def __init__(self, capacity):
        self.__capacity = capacity
        self.__stack = []

    def push(self, value):
        self.__stack.append(value)

    def print(self):
        print(self.__stack)
```

```python
stack = MyStack(5)
stack.push(12)
stack.push(8)
stack.push(21)

stack.print()
```

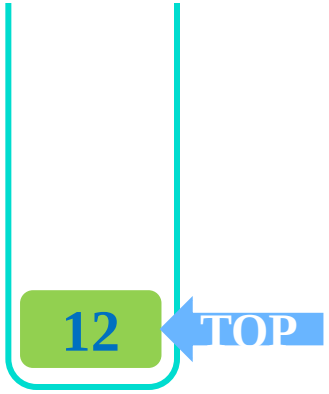
```
[12, 8, 21]
```

Stack Data Structure

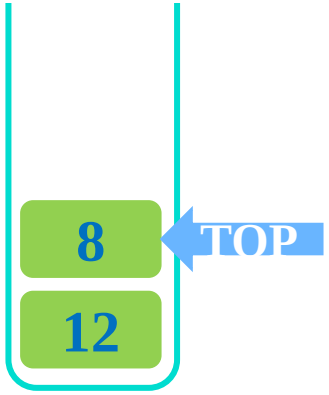
STEP 1: Create Empty Stack



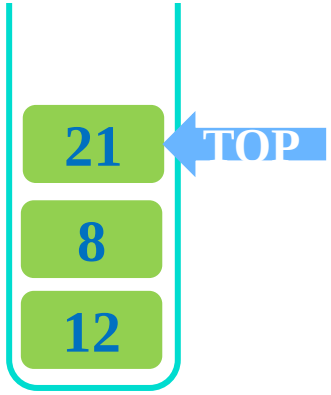
STEP 2: PUSH(12)



STEP 3: PUSH(8)



STEP 3: PUSH(21)



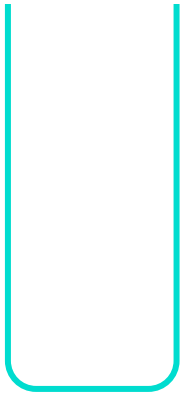

```python
class MyStack:
    def __init__(self, capacity):
        self.__capacity = capacity
        self.__stack = []
    def is_full(self):
        if len(self.__stack) == self.__capacity:
            return True
        else:
            return False
    def push(self, value):
        if self.is_full():
            print('Do nothing!')
        else:
            self.__stack.append(value)
    def print(self):
        print(self.__stack)
```

```python
stack = MyStack(5)
stack.push(12)
stack.push(8)
stack.push(21)
stack.print()
```

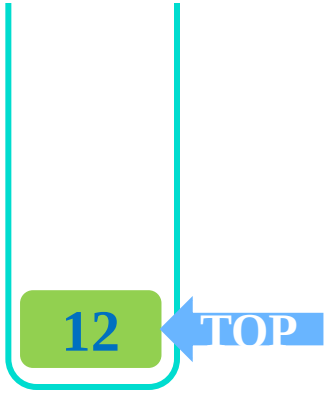
```
[12, 8, 21]
```

Stack Data Structure

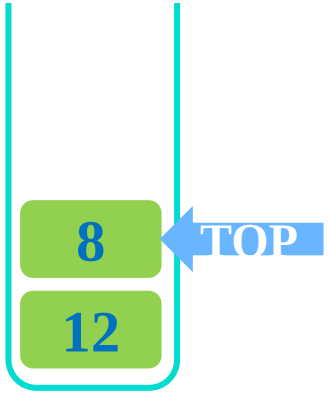
STEP 1: Create Empty Stack



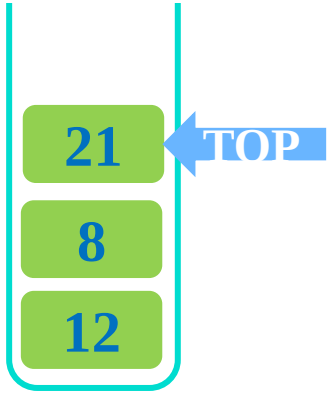
STEP 2: PUSH(12)



STEP 3: PUSH(8)



STEP 3: PUSH(21)

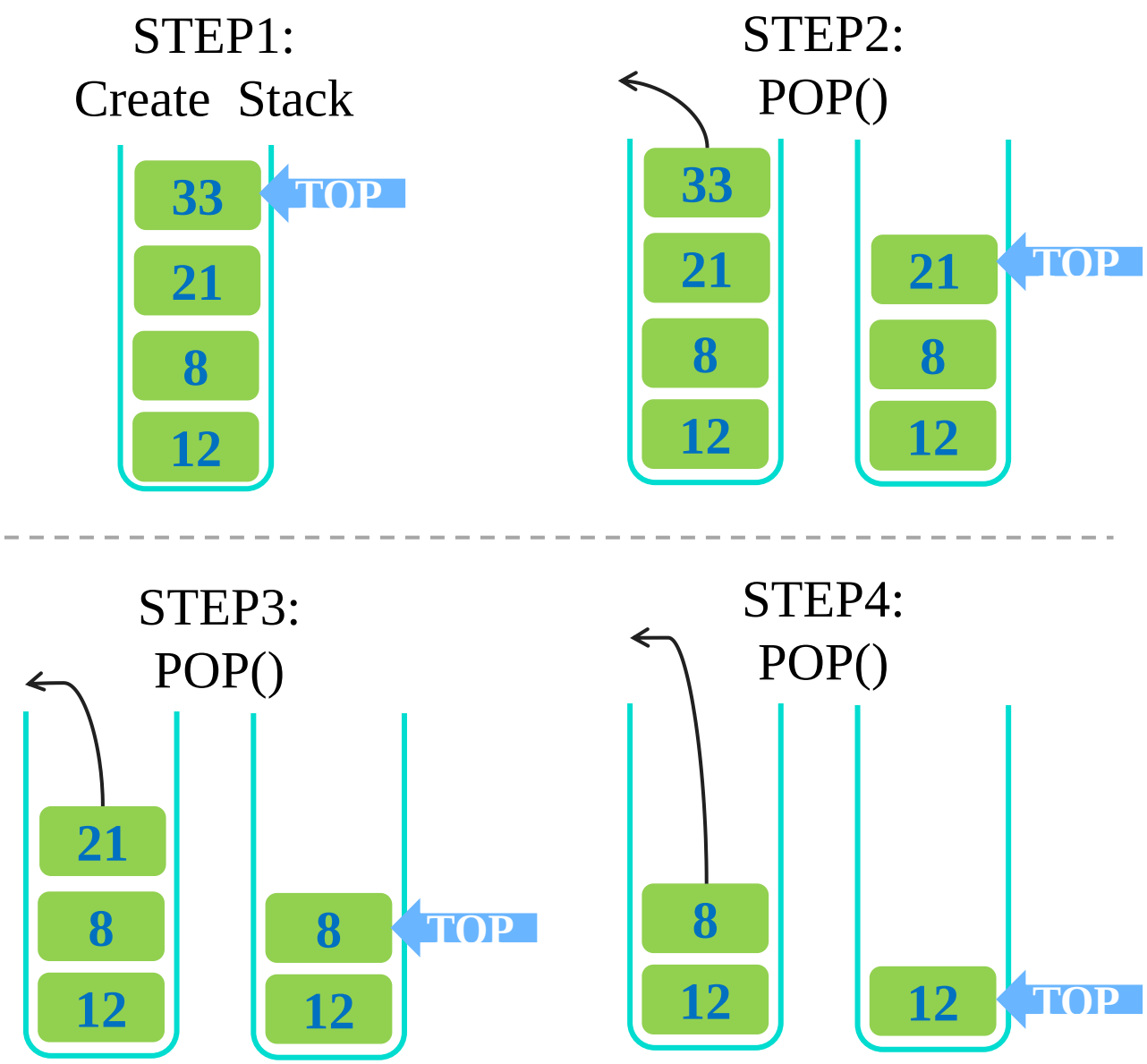



```python
class MyStack:
    def __init__(self, capacity):
        self.__capacity = capacity
        self.__stack = []
    def is_full(self):
        return len(self.__stack) == self.__capacity
    def push(self, value):
        if self.is_full():
            print('Do nothing!')
        else:
            self.__stack.append(value)
    def print(self):
        print(self.__stack)
```

```python
stack = MyStack(5)
stack.push(12)
stack.push(8)
stack.push(21)
stack.push(33)
stack.push(34)
stack.push(35)
stack.print()
```

```
Do nothing!
[12, 8, 21, 33, 34]
```

Stack Data Structure




```python
class MyStack:
    def __init__(self, capacity):
        self.__capacity = capacity
        self.__stack = []
    def is_empty(self):
        return len(self.__stack) == 0
    def pop(self):
        if self.is_empty():
            print('Do nothing!')
            return None
        else:
            return self.__stack.pop()
    def print(self):
        print(self.__stack)
```

```python
stack = MyStack(5)
# ...
stack.print()
stack.pop()
stack.pop()
stack.pop()
stack.print()
```

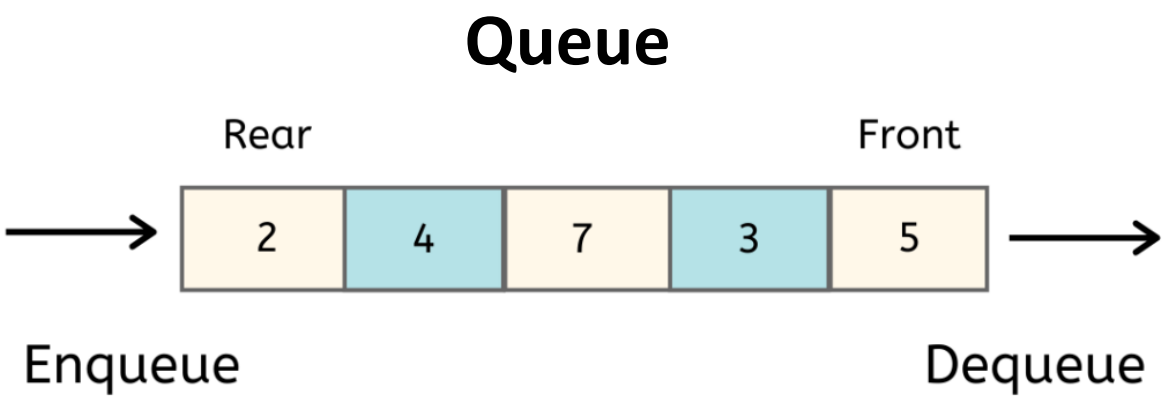
```
[12, 8, 21, 33]
[12]
```

Queue Data Structure

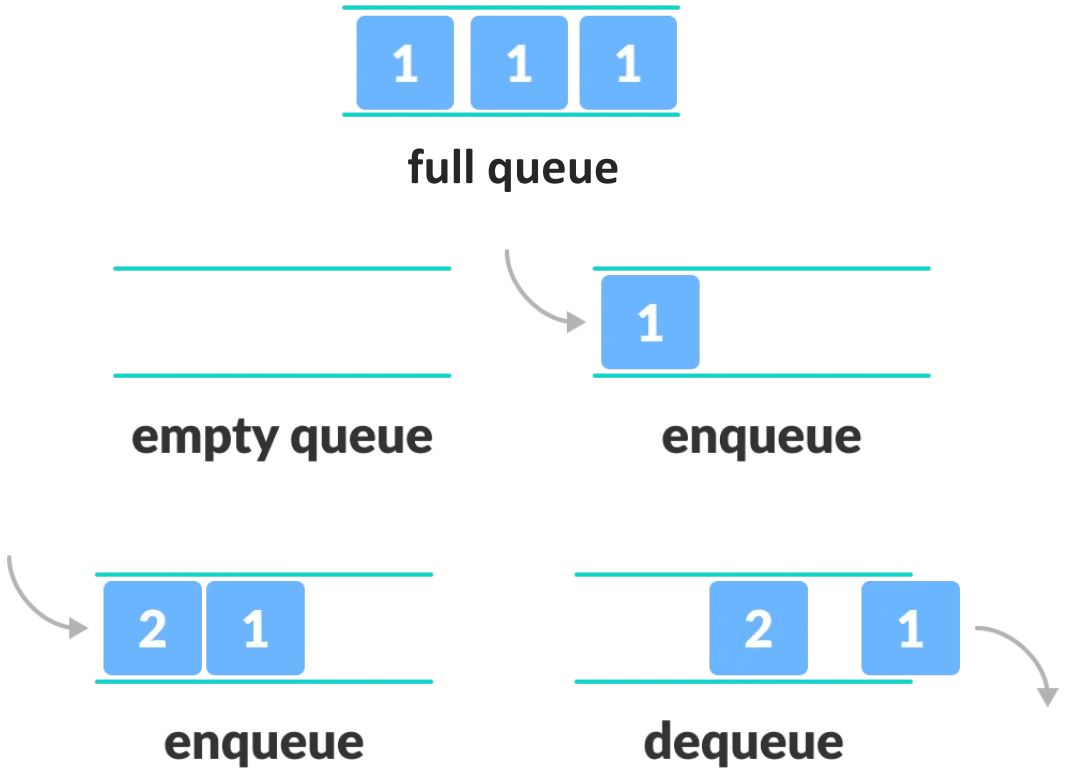
❖ Introduction

- Operate in a **FIFO (First in First out)** context
- **Element** are **inserted** (enqueue) and **extracted** (dequeue) happens at **OPPOSITE** ends

Basic Operations of Queue
- **enqueue**: Add an element to the end of the queue
- **dequeue**: Remove an element from the front of the queue
- **is_empty**: Check if the queue is empty
- **is_full**: Check if the queue is full
- **peek**: Get value of the front of the queue without removing it



First in First out
(FIFO)



List Review

❖ Add an element

data =

6	5	7	1	9	2

data.append(4) # thêm 4 vào vị trí cuối list

data =

6	5	7	1	9	2	4

```
1 data = [6, 5, 7, 1, 9, 2]
2 print(data)
3 data.append(4)
4 print(data)
```

```
[6, 5, 7, 1, 9, 2]
[6, 5, 7, 1, 9, 2, 4]
```

❖ Deleting an element

data =

6	5	7	1	9	2

data.pop(0) # xóa phần tử ở vị trí đầu tiên

data =

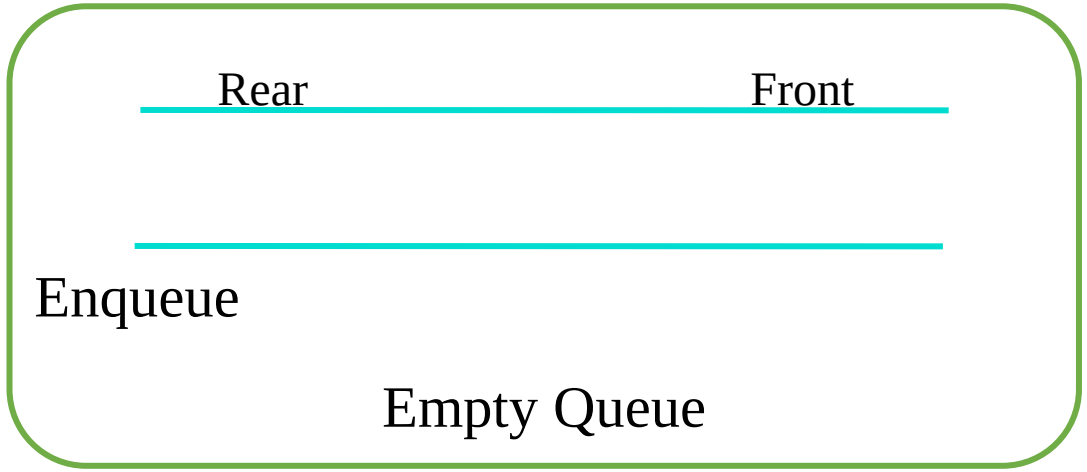
5	7	1	9	2

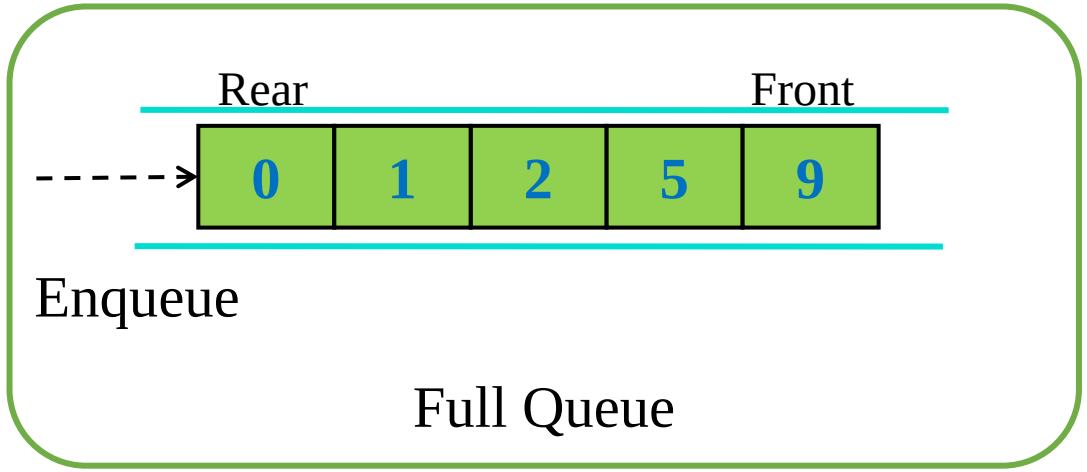
```
1 data = [6, 5, 7, 1, 9, 2]
2 print(data)
3 data.pop(0)
4 print(data)
```

```
[6, 5, 7, 1, 9, 2]
[5, 7, 1, 9, 2]
```

Queue Data Structure

❖ enqueue()





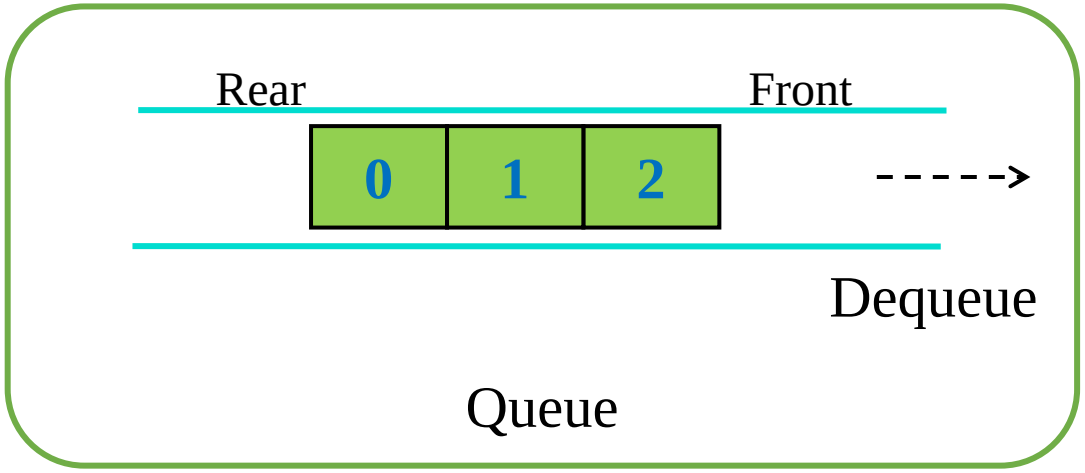

```python
class MyQueue:
    def __init__(self, capacity):
        self.__capacity = capacity
        self.__data = []

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

    def enqueue(self, value):
        if self.is_full():
            print('Do nothing!')
        else:
            self.__data.append(value)

    def print(self):
        print(self.__data)
```

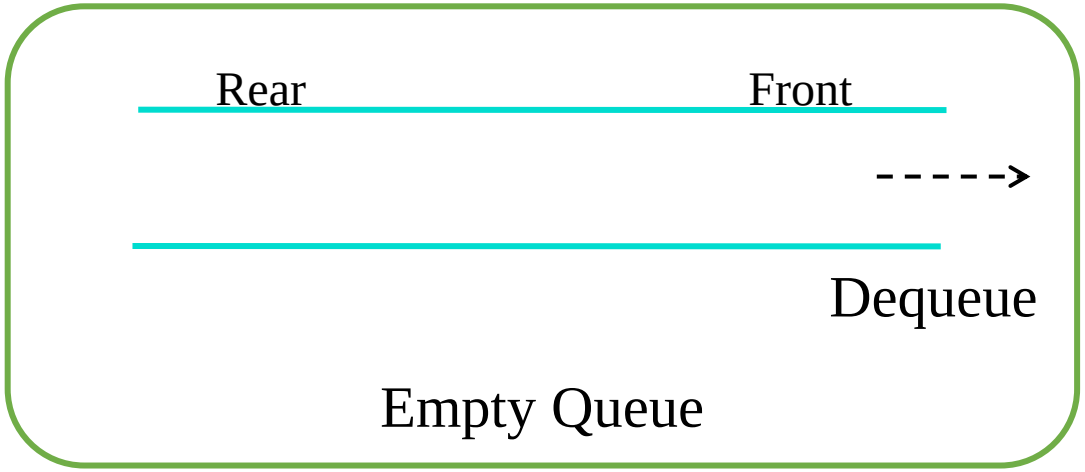
```python
queue = MyQueue(5)
queue.print()

queue.enqueue(9)
queue.enqueue(5)
queue.enqueue(2)
queue.enqueue(1)
queue.enqueue(0)
queue.enqueue(6)
queue.print()
```

```
[]
Do nothing!
[9, 5, 2, 1, 0]
```

Queue Data Structure

❖ dequeue()





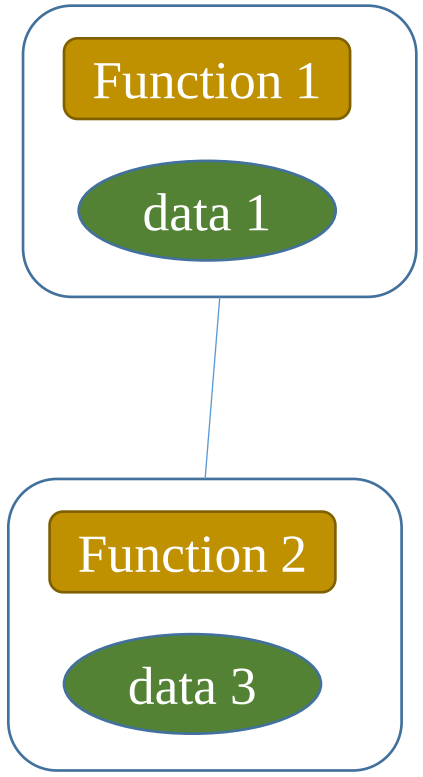

```python
class MyQueue:
    def __init__(self, capacity):
        self.__capacity = capacity
        self.__data = []

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

    def dequeue(self):
        if self.is_empty():
            print('Do nothing!')
            return None
        else:
            return self.__data.pop(0)

    def print(self):
        print(self.__data)
```

```python
queue = MyQueue(5)
#...
queue.print()

queue.dequeue()
queue.dequeue()
queue.dequeue()
queue.print()
```

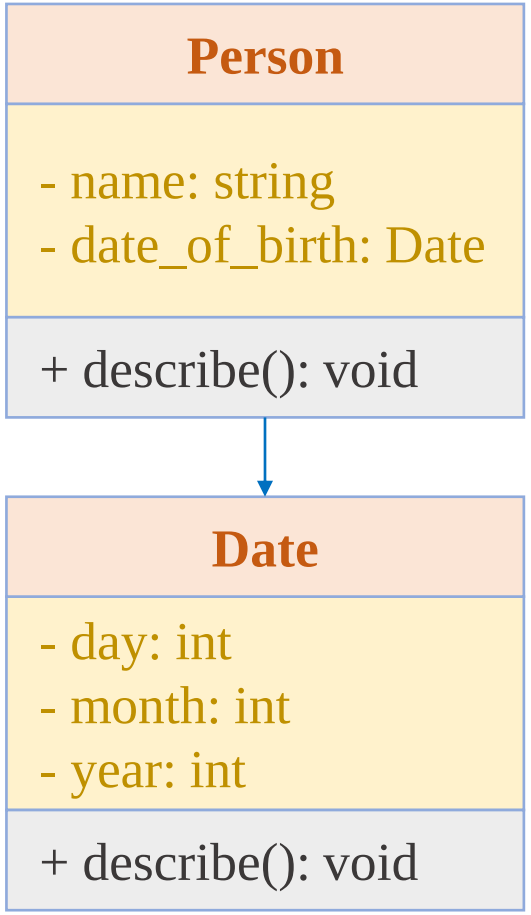
```
[2, 1, 0]
[]
```

Summary

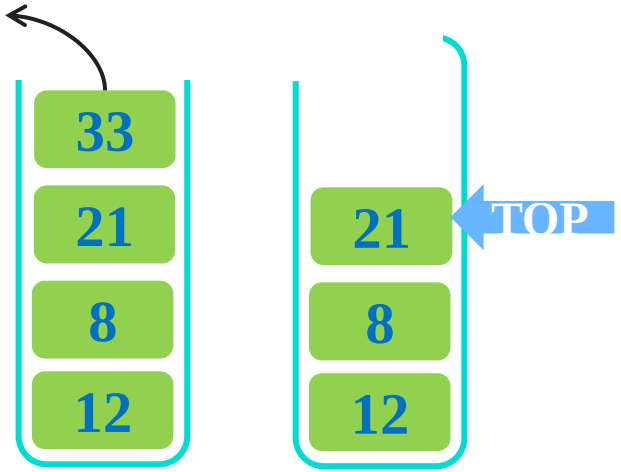
Class (Encapsulation)

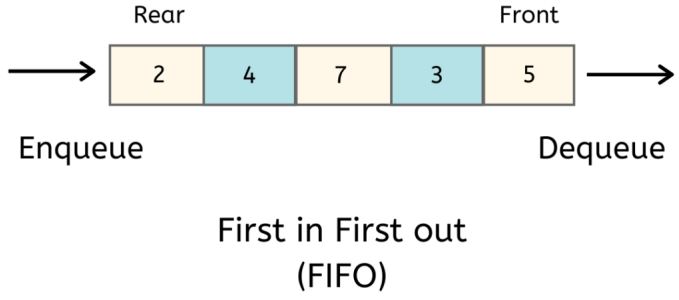


Delegation



Stack & Queue





First in First out
(FIFO)