OBJECT-ORIENTED PROGRAMMING

TUAN NGUYEN

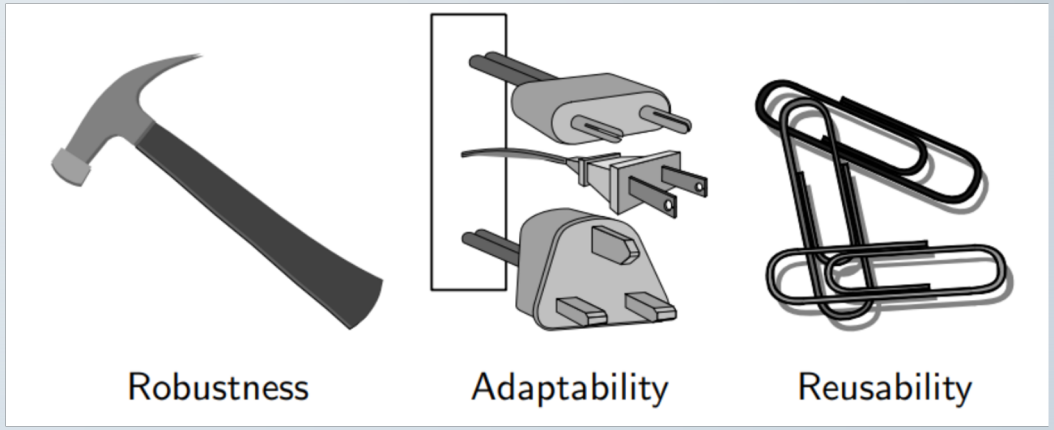
OUTLINE

- Object-oriented design
- Object vs class
- Python class
- Operator overloading (special method)
- Inheritance

Object-Oriented Design Goal

Software implementations should achieve ***robustness***, ***adaptability***, and ***reusability:***

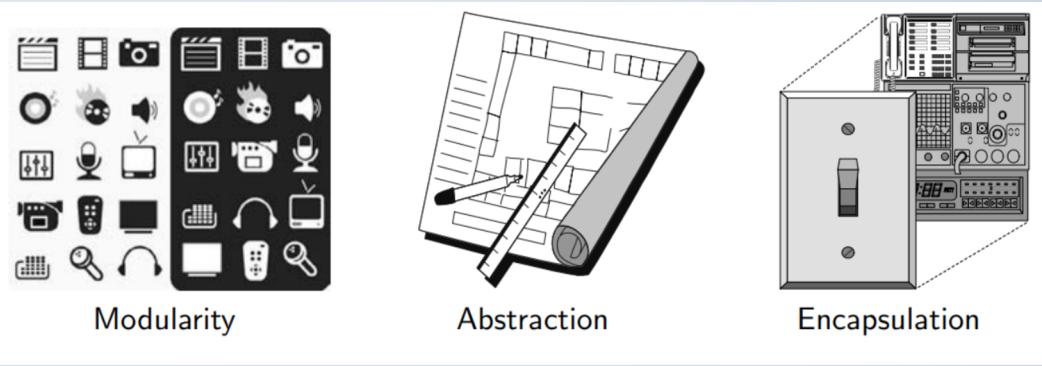
- ***Robustness:*** handle unexpected input
- ***Adaptability:*** minimal change on different hardware and operating system platforms
- ***Reusability:*** reusable code in different applications.



Object-Oriented Design Principles

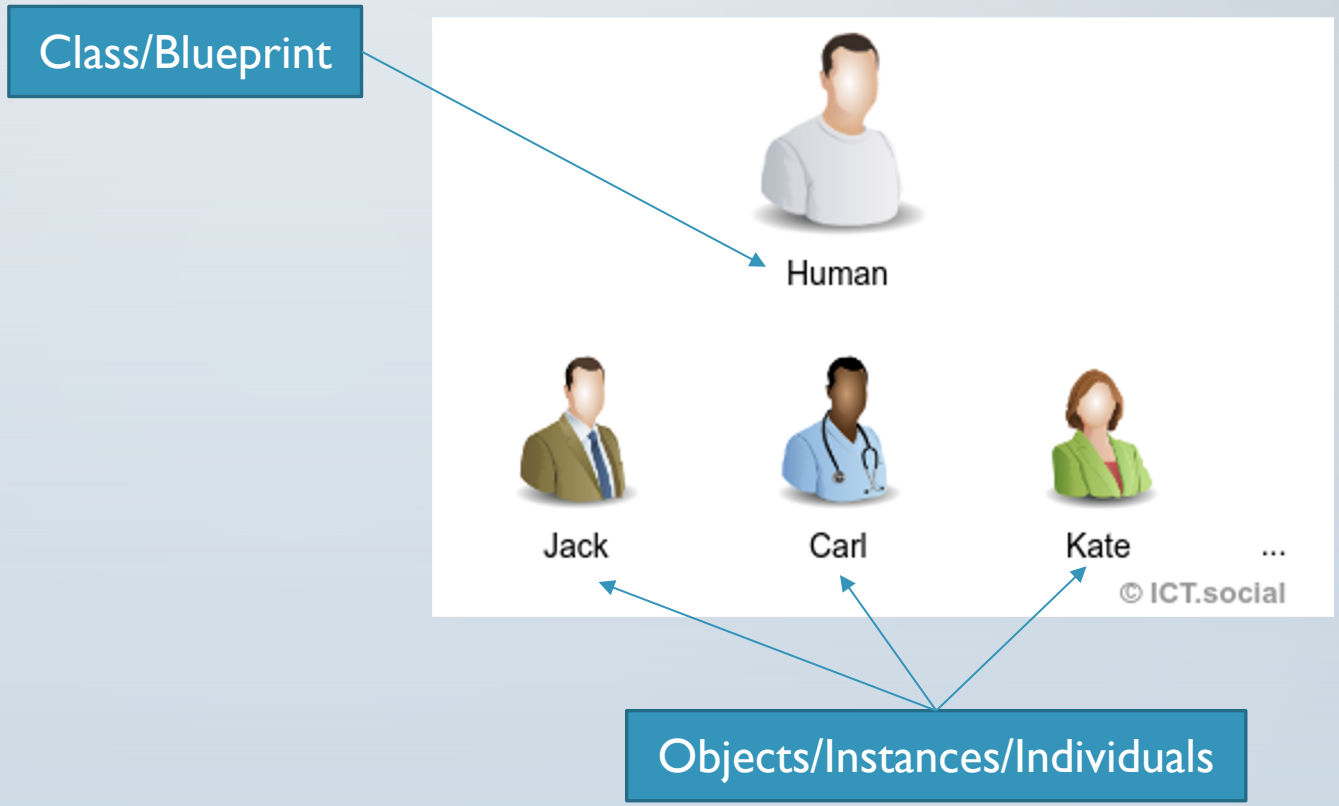
Principles of the object-oriented approach:

- Modularity
- Abstraction
- Encapsulation



Class

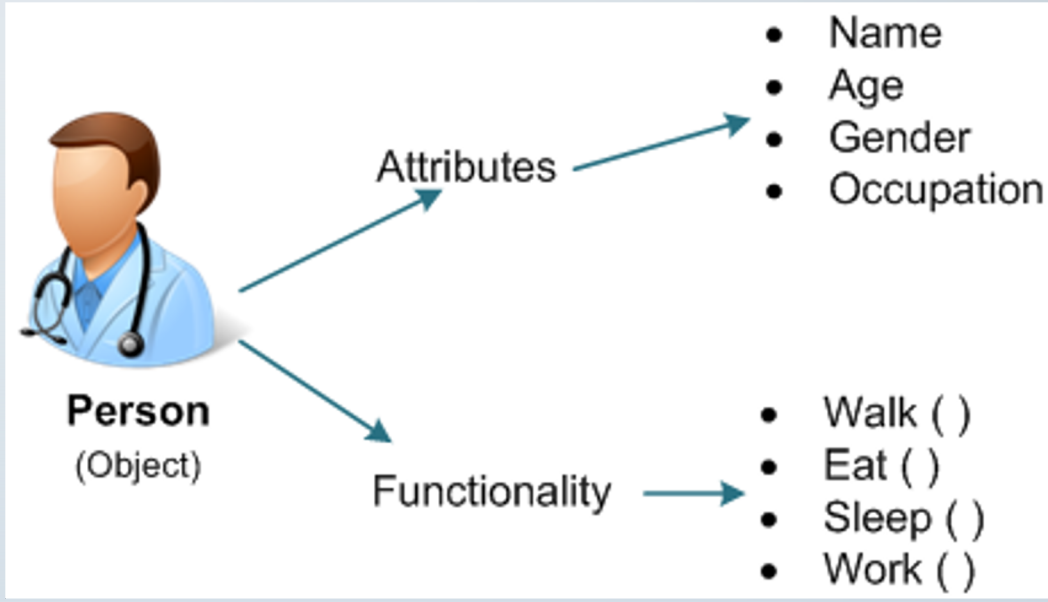
Class is a specification of an object or template for creation of object.



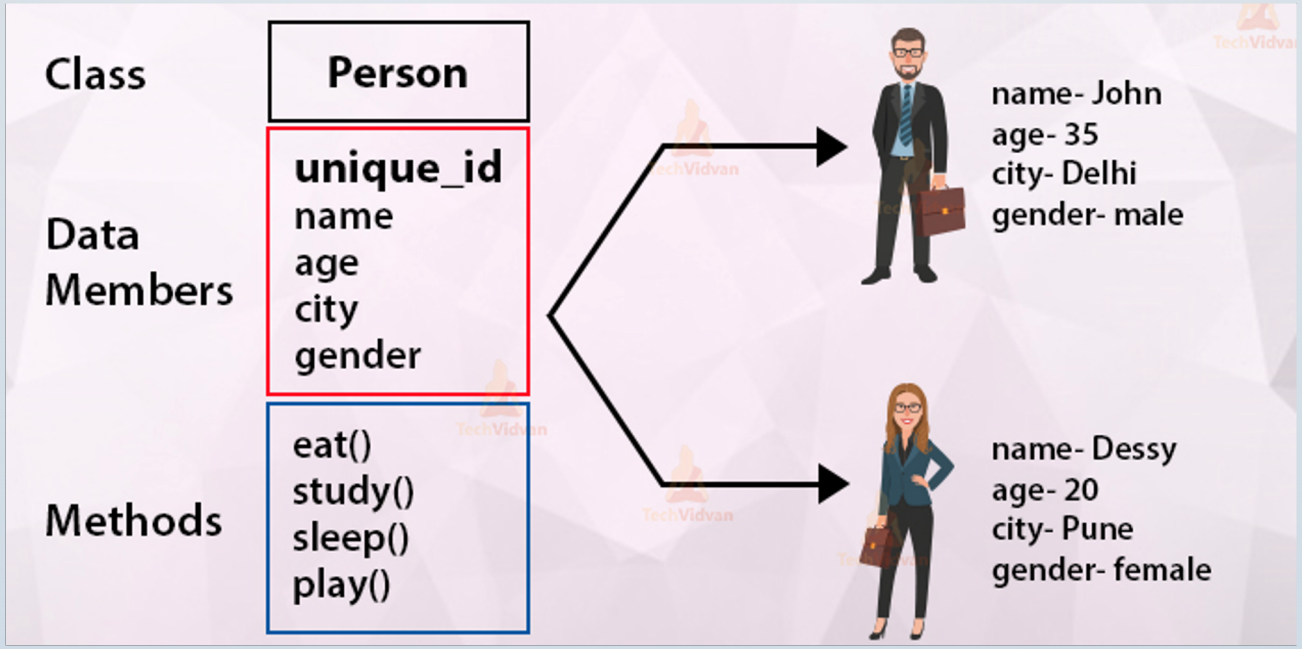
Object

The object is the real-life thing which has characteristics and behaviors.

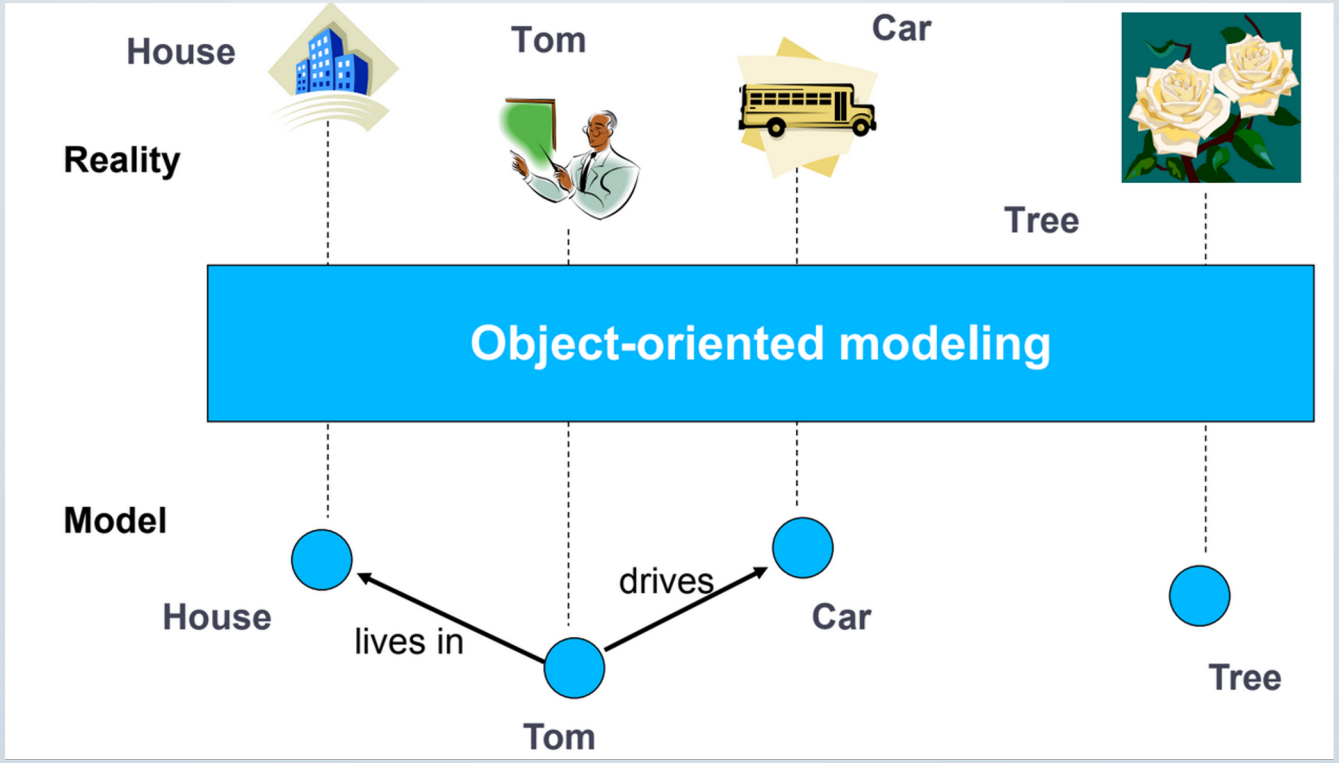
In programming that called attributes (variables/data members) and methods (functions).



Class vs Object



OOP World

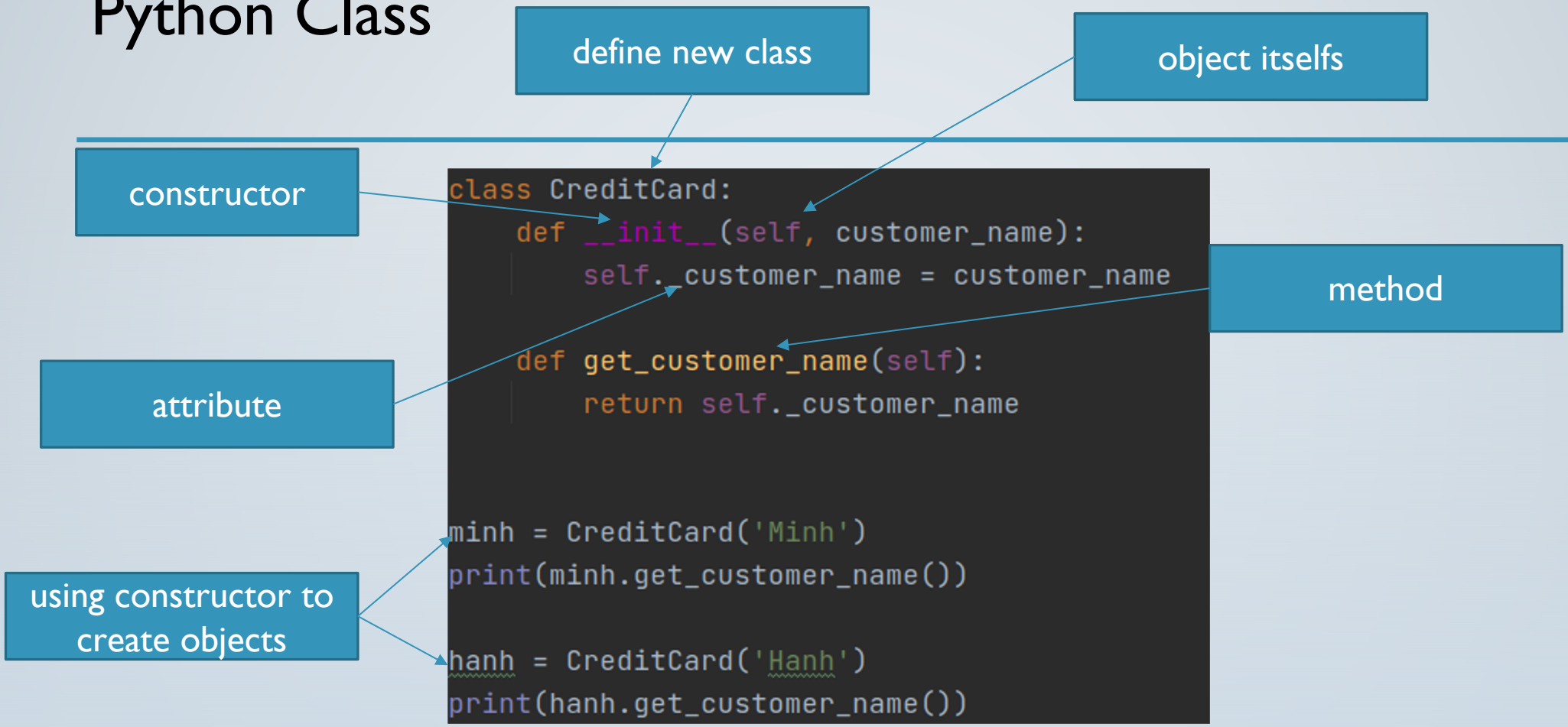

House
Tom
Car
Reality
Tree
Object-oriented modeling
Model
House
lives in
drives
Car
Tom
Tree

Object Communication

Communication in the real world

Communication in the OOP world

Python Class



Python Class (1)

```
class CreditCard:
    def __init__(self, customer_name, balance):
        self._customer_name = customer_name
        self._balance = balance

    def get_customer_name(self):
        return self._customer_name

    def get_balance(self):
        return self._balance

    def make_payment(self, amount):
        self._balance -= amount


minh = CreditCard('Minh', 100)
minh.make_payment(50)
print(minh.get_balance())
minh.make_payment(20)
print(minh.get_balance())
```

_customer_name = ‘Minh’
_balance = 100

_customer_name = ‘Minh’
_balance = 50

_customer_name = ‘Minh’
_balance = 30

Case Study: Credit Card

Class:	CreditCard	
Fields:	_customer	_balance
	_bank	_limit
	_account	
Behaviors:	get_customer( )	get_balance( )
	get_bank( )	get_limit( )
	get_account( )	charge(price)
	make_payment(amount)	

Write python code to implement the CreditCard class

Isinstance

Everything in Python is object, including: integer, function, list,...

=> how does Python understand the add of two numbers (objects): a + b.

```
def hello():
    print('hello')


minh = CreditCard('Minh', 100)
a = 5

print(isinstance(a, object))  # True
print(isinstance(a, int))  # True
print(isinstance(a, float))  # False

print(isinstance(hello, object))  # True
print(isinstance(minh, CreditCard))  # True
print(isinstance(minh, int))  # False
```

Operator Overloading

+ operator is overloaded by implementing a method named __add__

a + b is similar to a.__add__(b)

```
class CreditCard:
    def __init__(self, customer_name, balance):
        self._customer_name = customer_name
        self._balance = balance

    def get_balance(self):
        return self._balance

    def make_payment(self, amount):
        self._balance -= amount

    def __add__(self, other):
        return self._balance + other.get_balance()


minh = CreditCard('Minh', 100)
hung = CreditCard('Hung', 50)

print(minh + hung)
```

Special Method

Common Syntax	Special Method Form	
a + b	a.__add__(b);	alternatively b.__radd__(a)
a − b	a.__sub__(b);	alternatively b.__rsub__(a)
a * b	a.__mul__(b);	alternatively b.__rmul__(a)
a / b	a.__truediv__(b);	alternatively b.__rtruediv__(a)
a // b	a.__floordiv__(b);	alternatively b.__rfloordiv__(a)
a % b	a.__mod__(b);	alternatively b.__rmod__(a)
a ** b	a.__pow__(b);	alternatively b.__rpow__(a)
a << b	a.__lshift__(b);	alternatively b.__rlshift__(a)
a >> b	a.__rshift__(b);	alternatively b.__rrshift__(a)
a & b	a.__and__(b);	alternatively b.__rand__(a)
a ^ b	a.__xor__(b);	alternatively b.__rxor__(a)
a | b	a.__or__(b);	alternatively b.__ror__(a)
a += b	a.__iadd__(b)	
a −= b	a.__isub__(b)	
a *= b	a.__imul__(b)	
...	...	
+a	a.__pos__()	
−a	a.__neg__()	
~a	a.__invert__()	
abs(a)	a.__abs__()	

Special Method (1)

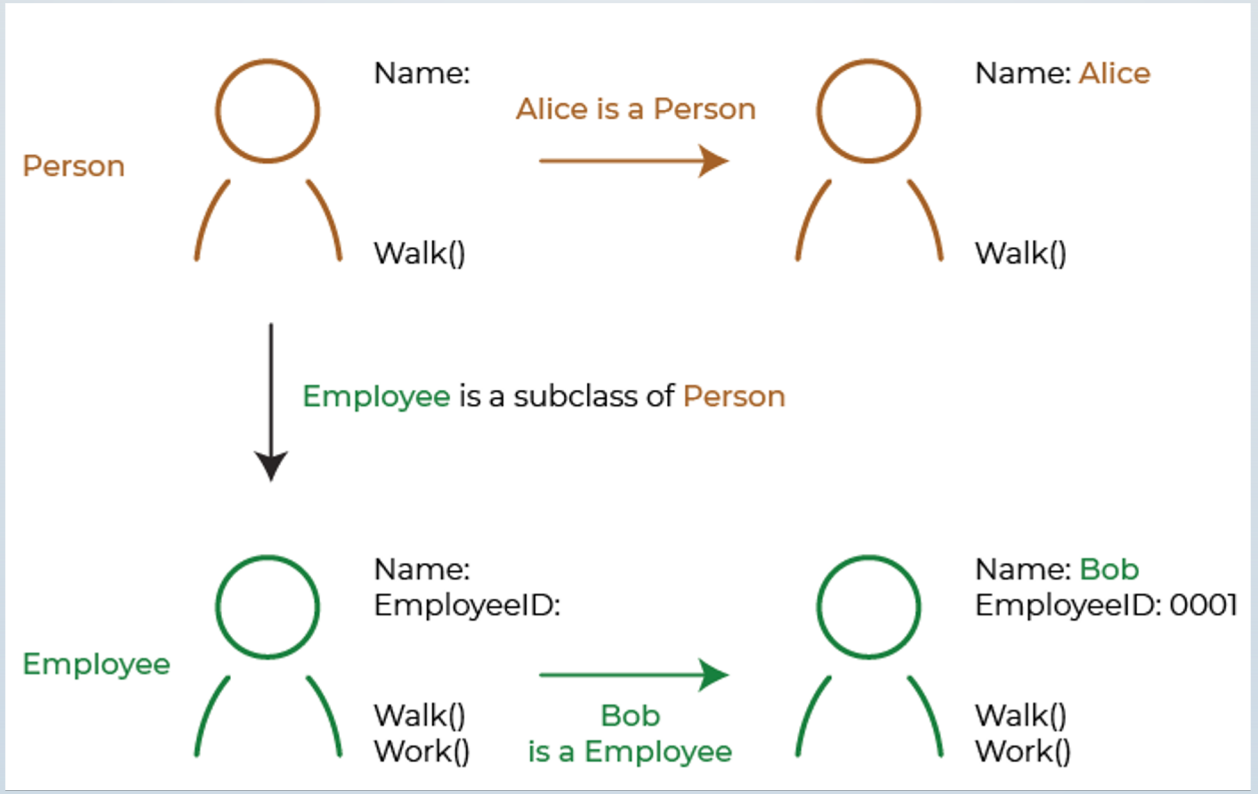
a < b	a.__lt__(b)
a <= b	a.__le__(b)
a > b	a.__gt__(b)
a >= b	a.__ge__(b)
a == b	a.__eq__(b)
a != b	a.__ne__(b)
v in a	a.__contains__(v)
a[k]	a.__getitem__(k)
a[k] = v	a.__setitem__(k,v)
del a[k]	a.__delitem__(k)
a(arg1, arg2, ...)	a.__call__(arg1, arg2, ...)
len(a)	a.__len__()
hash(a)	a.__hash__()
iter(a)	a.__iter__()
next(a)	a.__next__()
bool(a)	a.__bool__()
float(a)	a.__float__()
int(a)	a.__int__()
repr(a)	a.__repr__()
reversed(a)	a.__reversed__()
str(a)	a.__str__()

Exercise

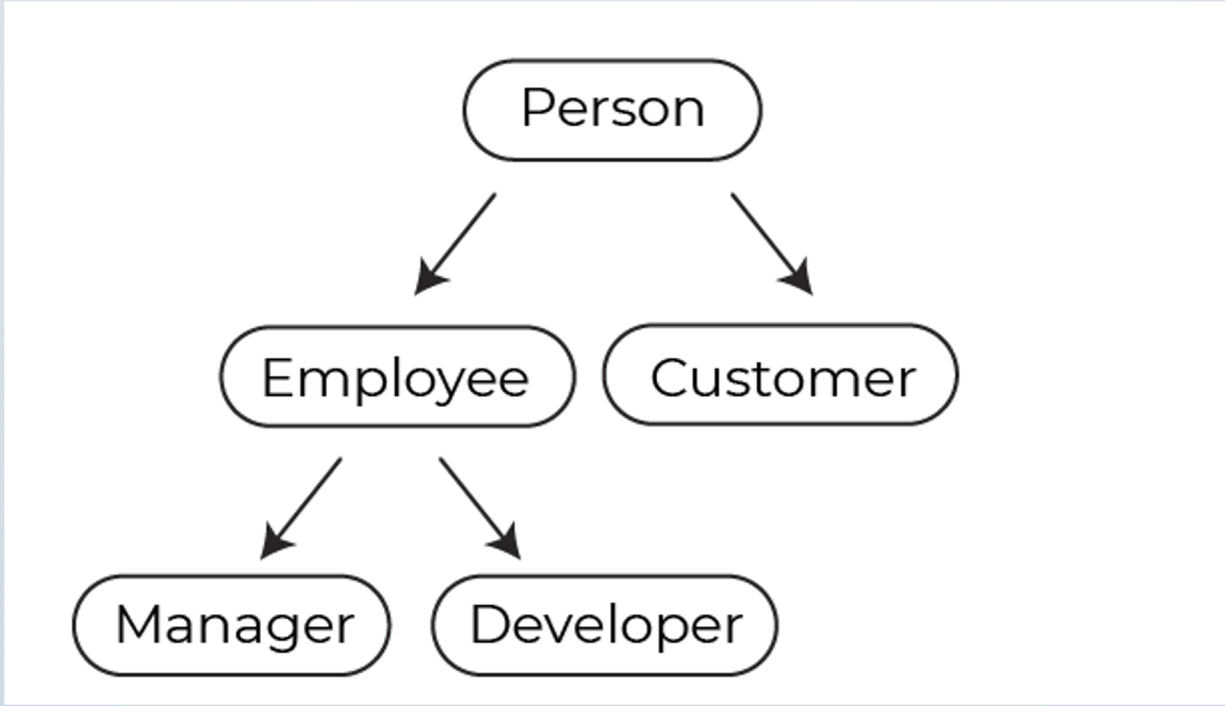
Write a 2D vector class:

- Attributes: x, y
- Special methods: add, substract, multiply by a scalar
- Other methods: dot product of two vector

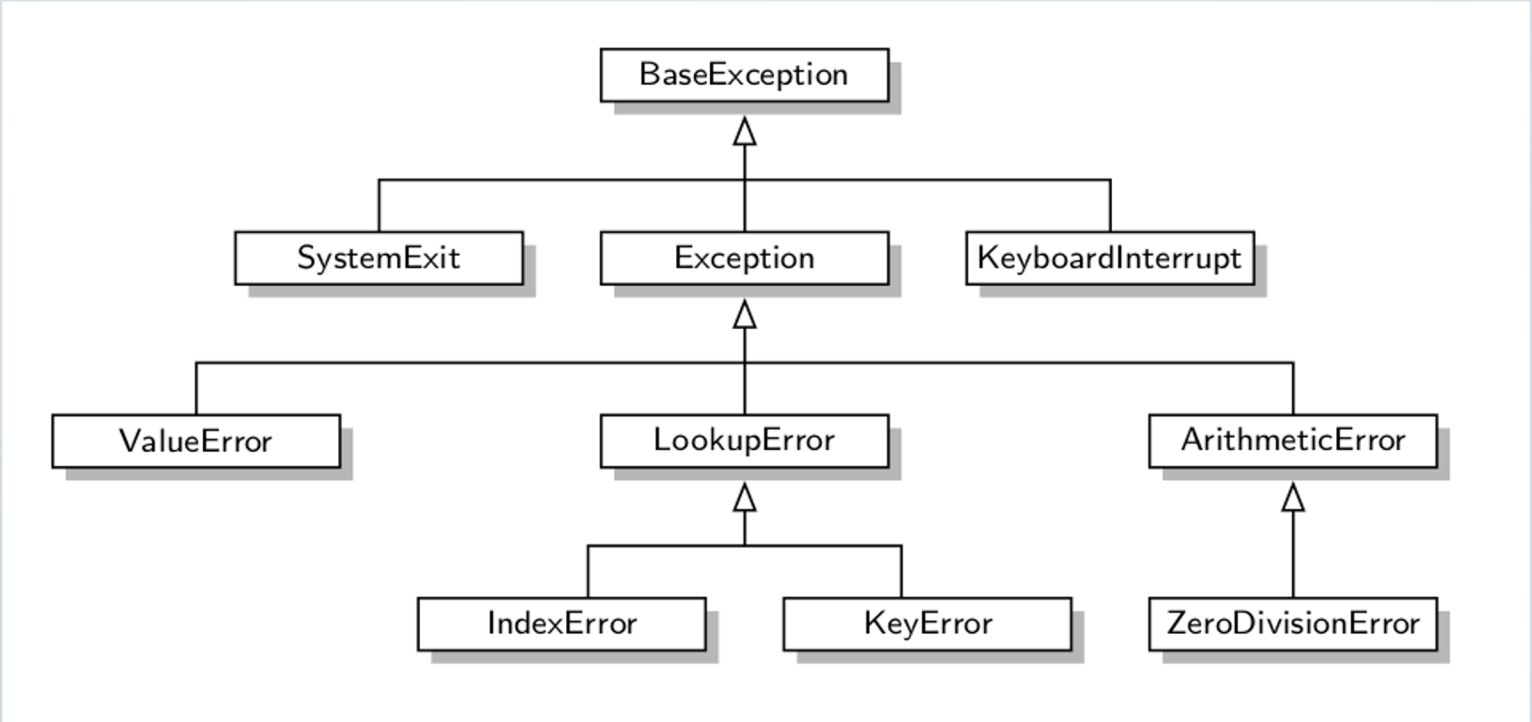
Inheritance



Inheritance (1)



Python's Exception Hierarchy



Python Inheritance

Employee class inherits from Person class

Parent's class

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

    def walk(self):
        print('person walk')


class Employee(Person):
    def __init__(self, name, id):
        super().__init__(name)
        self._id = id

    def walk(self):
        super().walk()
        print('employee walk')


emp = Employee('Minh', 'E01')
emp.walk()
```

Case Study: PredatoryCreditCard

The PredatoryCreditCard differ CreditCard:

- If an attempted charge is rejected because it would have exceeded the credit limit, a $5 fee will be charged
- There will be a mechanism for assessing a monthly interest charge on the outstanding balance, based upon an AnnualPercentage Rate (APR) specified as a constructor parameter