Maktab Sharif

Python | Main course

Session 5

- OOP intro
- OOP in python
- Class vs Instance (Methods & Attrs)
- Static Methods

by Mohammad Amin H.B. Tehrani - Reza Yazdani

www.maktabsharif.ir

Review

Contents

Maktab Sharif

1) Variables
2) Types
3) Operators
4) String
5) Data structures (List, Dict, Tuple, Set)
6) Conditional Statements (if .. else ..)
7) Loops (While, For)
8) Function
9) Built-in functions: map, sorted, filter, …
10) List comprehension (inline for), Trenary expression (inline if), lambda

Object-Oriented Programing
Introduction

Maktab Sharif

OOP philosophy

Maktab Sharif

Object-oriented programming (**OOP**) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods)

You should divide your program not into tasks,
but into models of physical objects.

Procedural programming
What does this program do?

vs.

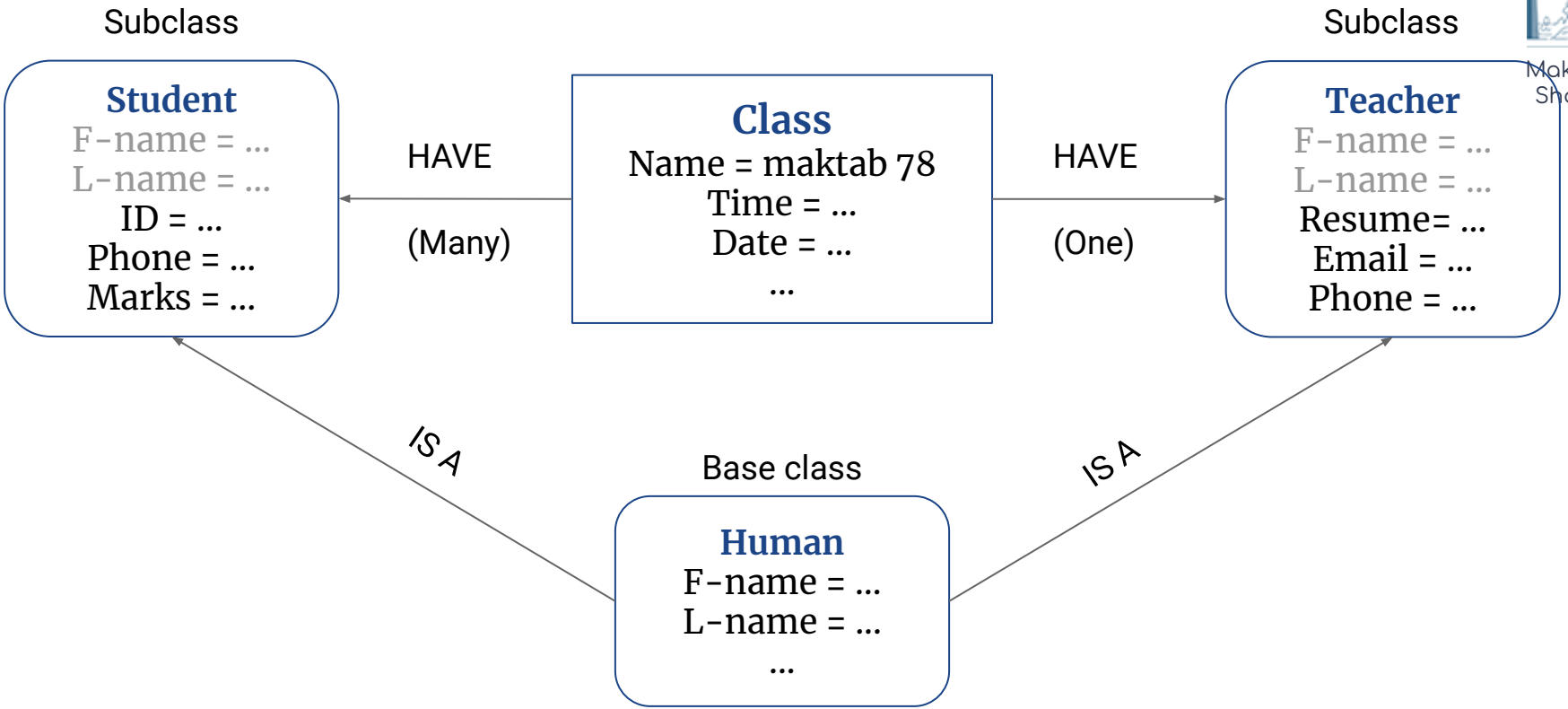
Object-oriented programming
What real world objects am I modeling?

Object-Oriented Fundamentals

Maktab Sharif

Principles:

1) Class
2) Object
3) Hierarchy
4) Encapsulation
5) Abstraction
6) Inheritance
7) Polymorphism



Example: Maktab 78 class!

Class

Maktab Sharif

A user-defined prototype for an object that defines a set of attributes that characterize any object of the class.

➔ Create a class, which is like a **blueprint** for creating an object

Syntax

```
class ClassName:
    ...
```

```
class Square:
    x = 10
    y = 20
    ...
```

```
class Student:
    name = 'Akbar'
    marks = []
    ...
```

Instantiate an Object in Python

Maktab Sharif

Instance: An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle

Creating a new object from a class is called instantiating an object. You can instantiate a new object by typing the name of the class, followed by opening and closing parentheses:

Syntax

`ins = ClassName(...)`

```
class Square:
    x = 10
    y = 20
    ...

s = Square()
```

```
class Student:
    name = 'Akbar'
    marks = []
    ...

S = Student()
```

Instance/Object Attributes (fields)

Maktab Sharif

An **instance/object** attribute is a variable that belongs to one (and only one) object. Every instance of a class points to its own attributes variables.

```
class Human:
    first_name = ...
    last_name = ...
    age: int
    gender: str
    height: int
    ...
```

```
class Car:
    brand: str

    def __init__(self):
        self.model = ...
        self.color = ...
        self.fuel = ...
```

Instance/Object Methods

Maktab Sharif

Methods are functions defined inside the body of a class. They are used to define the **behaviors** of an object.
A method is a function that "belongs to" an object.

```
class Human:
    name = ...

    def sleep(self, time):
        ...

    def eat(self, food):
        ...
```

```
class Car:
    speed = ...

    def start(self):
        ...

    def brake(self):
        ...
```

Initialize object (Constructor)

Maktab Sharif

Method: __init__(self, ...)

__init__ is one of the reserved methods in Python. In object oriented programming, it is known as a constructor. The __init__ method can be called when an object is created from the class, and access is required to initialize the attributes of the class.

```
class Human:

    def __init__(self, first_name, last_name, **extra_information):
        self.name = first_name + last_name
        self.extra_info = extra_information


akbar = Human('Akbar', 'Rezaii', age=25, height=168)
```

Example

Maktab Sharif

```
class Square:

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def area(self):
        return self.x * self.y


s = Square(2, 5)
print(s.area())
```

Self keyword

Maktab Sharif

The **self** parameter is a reference to the **current instance** of the class, and is used to access variables that belongs to the class.
It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class

```
class MyClass:

    def __init__(self, a, ...):
        self.my_attr = a
        self.my_method()
        ...

    def my_method(my_self, ... ):  # Call as my_self
        my_self...
```

Class vs. Instance (methods & attrs)

Class vs. Instance attributes

Maktab Sharif

- **An instance attribute** is a Python variable **belonging to one, and only one, object.** This variable is only accessible in the scope of this object and it is defined **inside** the constructor function, __init__(self,..) of the class.

- **A class attribute** is a Python variable that **belongs to a class rather than a particular object.** It is shared between all the objects of this class and it is defined **outside** the constructor function.

Example (1 of 3)

Maktab Sharif

Class definition:

```
class MyClass:
    class_attr = "It's a class Attribute!"

    def __init__(self, x=None):
        self.my_attr = x or "It's my Attribute!"

    def some_method(self):
        self.class_attr = "MY class_attr modified!"

    def another_method(self):
        MyClass.class_attr = "class_attr modified!"
```

Example (2 of 3)

Maktab Sharif

Instantiation:

```
ins1 = MyClass()
ins2 = MyClass("It's ins2 attribute!")

print(MyClass.class_attr, '', sep='\n')
print(ins1.class_attr, ins2.class_attr, '', sep='\n')

print(ins1.my_attr, ins2.my_attr, '', sep='\n')
print(id(MyClass.class_attr), id(ins1.class_attr), id(ins2.class_attr), end='\n\n')

ins1.some_method()
ins2.another_method()

print(ins1.class_attr, ins2.class_attr, '', sep='\n')
print(id(MyClass.class_attr), id(ins1.class_attr), id(ins2.class_attr), end='\n\n')

print(MyClass.my_attr)
```

Example (3 of 3)

Maktab Sharif

output:

```
It's a class Attribute!

It's a class Attribute!
It's a class Attribute!

It's my Attribute!
It's ins2 attribute!

2366069311088 2366069311088 2366069311088

MY class_attr modified!
class_attr modified!

2366069443360 2366069444160 2366069443360

AttributeError...
```

Static Methods

A **static** method does **not receive an implicit first argument.**

- A static method is also a method which is bound to the class and not the object of the class.
- A static method can't access or modify class state.
- It is present in a class because it makes sense for the method to be present in class.

Syntax:

```
@staticmethod
def method_name(...):
    ...
```

We generally use static methods to create utility functions.