

OOP in Python

Tran Giang Son, tran-giang.son@usth.edu.vn

ICT Department, USTH



Review



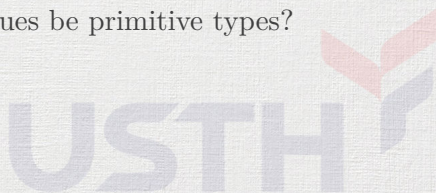
Questions!?!?!¹

- What is a class? What is an object?
- What is the difference between an object and a class?
- Where do objects come from (how do you create one)?
- How many objects of a given class can you have at a given time?
- Each data type in Java (and in many other languages) can be classified as one of two kinds. What are they, and how are they different?

¹Or exam?!

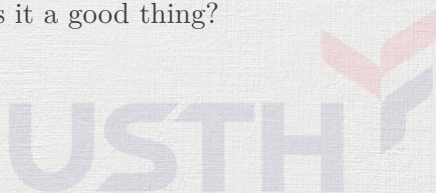
Questions!?!?!?

- What is a primitive type? What is a reference type (or object type)?
- What is a method?
- Can a class have more than one method with the same name? If so, are there any restrictions?
- What is a parameter? What is a return value?
- Can parameters and return values be primitive types? Reference types?



Questions!?!?!

- What is type matching or type conformance?
- What is a constructor?
- What is assignment? How is it different for reference types versus primitive types?
- What are accessor methods and mutator methods?
- What is abstraction and why is it a good thing?



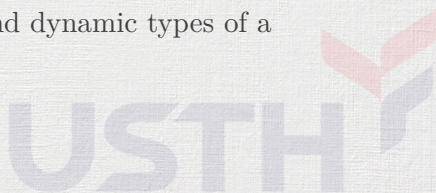
Questions!?!?!

- What is inheritance?
- What is an inheritance hierarchy?
- What is a subclass? A superclass?
- What are the advantages of using inheritance?
- What is the difference between an is-a and a has-a relationships?



Questions!?!?

- What is polymorphism?
- What are overriding and overloading? Are they the same? Give examples.
- What does the keyword super mean? When is it used?
- What does the keyword protected mean? When is it used, and what does it do?
- What is meant by the static and dynamic types of a variable?



Questions!?!?!

- What is an abstract class? When are they useful? Give an example.
- What is multiple inheritance? Why is it useful? Can it be done in Java? Is there a substitute for it?
- What is an interface? Why are they useful?



Object and Class



Previously, on PW #1

- Functions
 - Input functions:
 - Input number of students in a class
 - Input student information: id, name, DoB
 - Input number of courses
 - Input course information: id, name
 - Select a course, input marks for student in this course
 - Listing functions:
 - List courses
 - List students
 - Show student marks for a given course

Why

- Lists, dicts, tuples could be OOP'ed
 - Student
 - Course
 - StudentMark
- Easier to manage
- Close to real-world management



Why

Procedural Programming

- Variables and related functions are separated
- Programs is divided into functions

Object-Oriented Programming

- Variables and related functions are bound together
- Programs are divided into objects



How

- Define a class

```
class <ClassName>
```

- Define a method

```
def <methodName>([args])
```

- Define a constructor

```
def __init__([args])
```

- Create an object from class

```
<obj> = <ClassName>([args])
```

- self: current object



How

```
class Person:
    def print(self):
        print("Name:", self.name)
        print("Age:", self.age)

    def __init__(self, n, a):
        self.name = n
        self.age = a

macron = Person("Emmanuel Macron")
```



How

- Call an object's method
`<obj>.<methodName>([args])`
- Accessing an object's attribute
`<obj>.<attr> = "values"`



How

- Object comparison: `__lt__` method
 - Compares current object with another instance
 - Return True if less than² the other instance

```
def __lt__(self, other):  
    return self.age < other.age
```

²Hence its name is `__lt__`

How

- Object's string representation: `__str__` method
 - Defines how an object will be stringified
 - Mostly when using with `print()`

```
def __str__(self):  
    return f"My name is {self.name}. I am {self.age}."
```



How: complete example clazz.py

```
#!/usr/bin/env python3
class Person:
    def __init__(self, n, a):
        self.name = n
        self.age = a

    def describe(self):
        print("Name:", self.name)
        print("Age:", self.age)

    def __lt__(self, other):
        return self.age < other.age

    def __str__(self):
        return f"My name is {self.name}. I am {self.age}."

macron = Person("Emmanuel Macron", 43)
macron.describe()
print(macron)

biden = Person("Joe Biden", 78)
print(f"Macron is younger: {macron < biden}")
```


How: complete example

```
$ ./clazz.py
```

```
Name: Emmanuel Macron
```

```
Age: 43
```

```
My name is Emmanuel Macron. I am 43.
```

```
Macron is younger: True
```



Inheritance

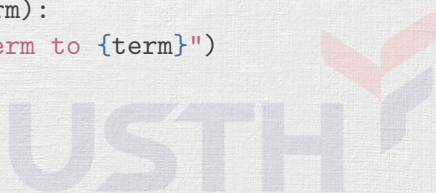


Defining Inheritance

- Define a child class with a superclass in parentheses
- All methods and attributes from superclass will be inherited to subclass

```
class Person:
    # already defined before...

class President(Person):
    def set_term(self, term):
        print(f"Setting term to {term}")
        self.term = term
```



Defining Inheritance

- Using the newly defined class and method

```
print("Macron is now President")
macron = President("Emmanuel Macron", 43)
macron.set_term(25)      # from President
macron.describe()       # from Person
```

```
$ ./clazz.py
Macron is now President
Setting term to 25
Name: Emmanuel Macron
Age: 43
```



Checking inheritance

- Built-in functions `isinstance()` and `issubclass()`
 - `isinstance()` returns True if the object is an instance of the class or other classes derived from it
 - `issubclass()` is used to check for class inheritance.



Checking inheritance

```
print(f"Macron is President: {isinstance(macron, President)}");  
print(f"Macron is Person: {isinstance(macron, Person)}");  
print(f"President is Person: {issubclass(President, Person)}");  
print(f"Person is President: {issubclass(Person, President)}");
```

```
$ ./clazz.py
```

```
Macron is President: True  
Macron is Person: True  
President is Person: True  
Person is President: False
```



Multiple Inheritance

- Python supports multiple inheritance
- Simply by adding more base class into the parentheses

```
class Person:  
    # already defined before...
```

```
class Employee:  
    def work(self):  
        print("I should be paid...")
```

```
class President(Person, Employee):  
    # already defined before...
```

Multiple Inheritance

```
print("Macron is now President")
macron = President("Emmanuel Macron", 43)
macron.set_term(25)      # from President
macron.describe()       # from Person
macron.work()            # from Employee
```

```
$ ./clazz.py
Setting term to 25
Name: Emmanuel Macron
Age: 43
I should be paid...
```



Polymorphism



Method overrides

- A superclass's method can be overridden, simply by **def**ing the same method name in the subclass
- A superclass instance can be accessed using `super()` in the subclass

```
class Person:
```

```
    # already defined before...
```

```
class President(Person, Employee):
```

```
    # something before
```

```
    def describe(self):
```

```
        super().describe()
```

```
        print("Term:", self.term)
```

```
    def work(self):
```

```
        super().work() # from Employee
```


Method overrides

- Using the overridden method

```
print("Macron is now President")
macron = President("Emmanuel Macron", 43)
macron.set_term(25)           # from President
macron.describe()             # from Person
macron.work()                  # from President
```

```
$ ./clazz.py
Setting term to 25
Name: Emmanuel Macron
Age: 43
I am well paid!
President is well paid!
```



Encapsulation



Private / Public access

- `public` by default
- No specified keyword
- Use underscore prefixes
 - `name`: public
 - `_name`: protected
 - `__name`: private



Private / Public access

- Accessor methods / Mutator methods
- Getter / Setter

```
class Employee:
    def __init__(self):
        self.__salary = 0

    def _get_salary(self):
        return self.__salary

    def set_salary(self, salary):
        self.__salary = salary

    def work(self):
        if self.__salary == 0:
            print("I should be paid...")
        else:
            print("I am well paid!")
```



Private / Public access

```
print("Macron is now President")
macron = President("Emmanuel Macron", 43)
macron.set_term(25)      # from President
macron.describe()       # from Person
macron.set_salary(1000) # from Employee
macron.work()           # from President
print(f"Macron salary is {macron._get_salary()}")
print(f"Macron salary is {macron.__salary}")

$ ./clazz.py
Macron is now President
Setting term to 25
Name: Emmanuel Macron
Age: 43
President is well paid!
Macron salary is 1000
Traceback (most recent call last):
  File ".../clazz.py", line 59, in <module>
    print(f"Macron's salary is {macron.__salary}")
AttributeError: 'President' object has no attribute '__salary'
```


Practical work 2: OOP'ed student mark management

- Copy your practical work 1 to 2.student.mark.oop.py
- Make it OOP'ed
- Same functions
 - Proper attributes and methods
 - Proper encapsulation
 - Proper polymorphism
 - e.g. `.input()`, `.list()` methods
- Push your work to corresponding forked Github repository