CSIT121
Object-Oriented Design and Programming

Dr. Fenghui Ren

School of Computing and Information Technology
University of Wollongong

Lecture 3 outline

- Python class and object
- Class implementation with Python
- Add attributes and methods
- Initialisation of objects
- Class vs instance variables
- Class vs instance vs static methods
- Public, protected and private variables and methods
- A case study

HelloWorld (OOD)

HelloWorld.py - /Users/fren/Library/CloudStorage/OneDrive-UniversityofWollongong/MyTeaching/...

```
class HelloWorld:

    def greet(self):
        print("Hello object world")


greeter = HelloWorld()
greeter.greet()
```

Ln: 1 Col: 0

Let's analyze the simplest possible program that has only one class and one method

Class definition

To define a class you need to:

1. Use the `class` keyword (spelled with all lowercase letters)
2. Specify its name. A class name is its identifier.

   By convention, class names
   - can include only letters, digits, and underscores. No spaces.
   - begin with a capital letter. If several words are combined, each shall starts with a capital letter too (CapWords notation)
   - should be meaningful `DpGh, Abc, R1`
     `WelcomeToJava, StaffMember, StreamBuffer`
3. The class definition line is followed by the class contents, indented.
4. Python uses four space indentation to delimit the classes, rather than brackets

Class definition

- Although our first class does nothing, we still can run the program.

1. Save the class definition in a file named HelloWorld.py
2. Implement and run the program in IDLE.

```
>>> class HelloWorld:
...     pass
...
>>> a=HelloWorld()
>>> b=HelloWorld()
>>> print(a)
<__main__.HelloWorld object at 0x101376d90>
>>> print(b)
<__main__.HelloWorld object at 0x101610d90>
```

Adding attributes

- Objects should contain data which are defined as the attribute
- We can set arbitrary attributes on an instantiated object using '.' (dot notation).
- Syntax: **<object>.<attribute> = <value>**

```
>>> a.firstWorld="Hello"
>>> a.secondWorld="World"
>>> b.firstWorld="Welcome"
>>> b.secondWorld=a.secondWorld
>>> print(a.firstWorld, a.secondWorld)
Hello World
>>> print(b.firstWorld, b.secondWorld)
Welcome World
```

Adding behaviors

- We have data, and it is time to add behaviors.
- We start with a method called ‘greet’ which prints “Hello object world”.
- ‘greet’ method does not require any parameters.
- We call the method through the dot notation: **<object>.<method>**

HelloWorld.py - /Users/fren/Library/CloudStorage/OneDrive-UniversityofWollongong/MyTeaching/...

```
class HelloWorld:

    def greet(self):
        print("Hello object world")


greeter = HelloWorld()
greeter.greet()
```

Ln: 1 Col: 0

Adding behaviors

- In Python, a method is defined with the **def** keyword, followed by a space, the name of the method, and a set of parentheses containing the parameter list.
- Method definition is terminated with a colon (:)
- The method body starts from the next line and is indented.

Syntax:

<def>< methodName><(parameter list):>

<four space indentation><method body>

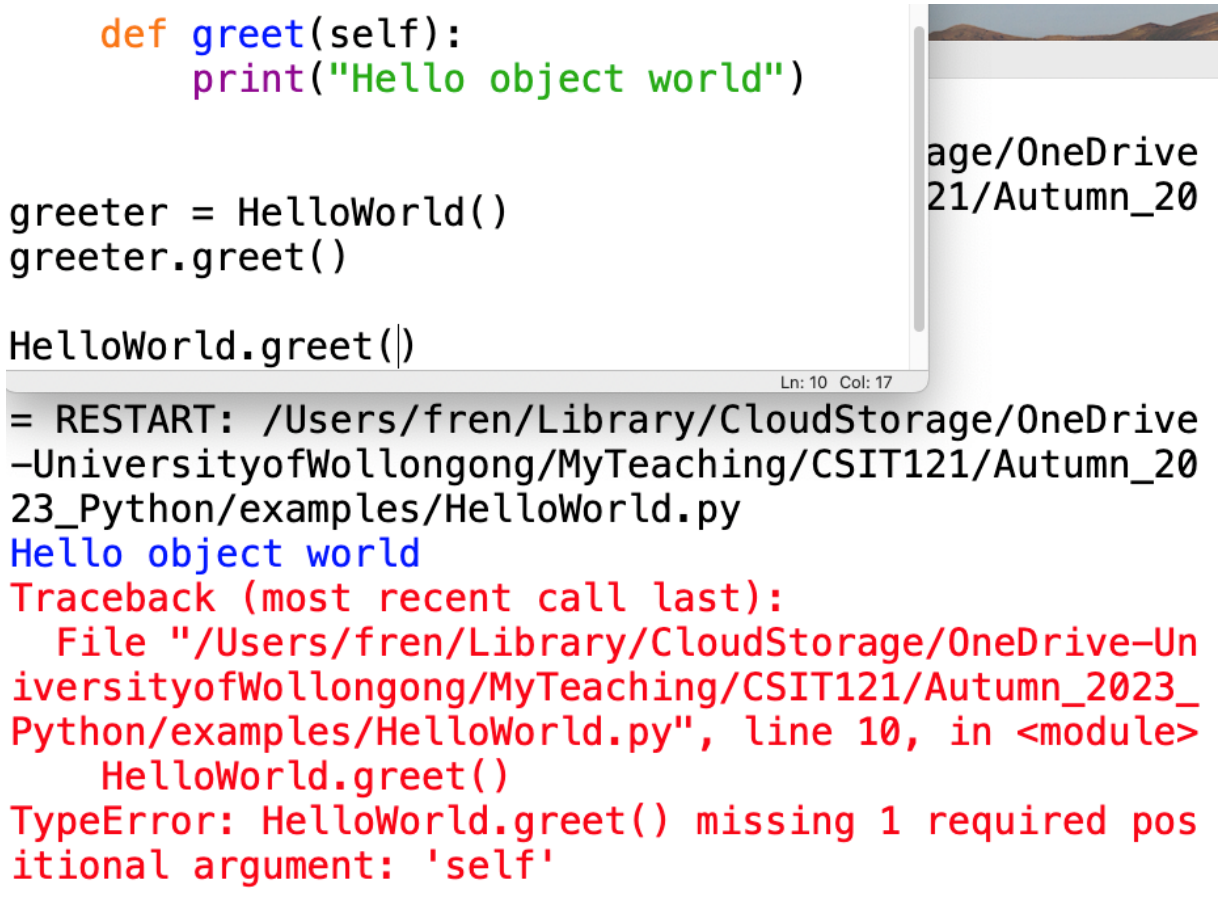
‘self’ argument

- Object methods shall contain an argument referred to the object itself.
- Normally, this argument is named ‘self’
- Of course, you can also call it with another name, such as ‘this’ or ‘bob’
- But we never seen a real Python programmer use any other name for this variable
- A normal function or a class method does not have this argument.
- The ‘self’ argument is a reference to the object that the method is being invoked on.

‘self’ argument

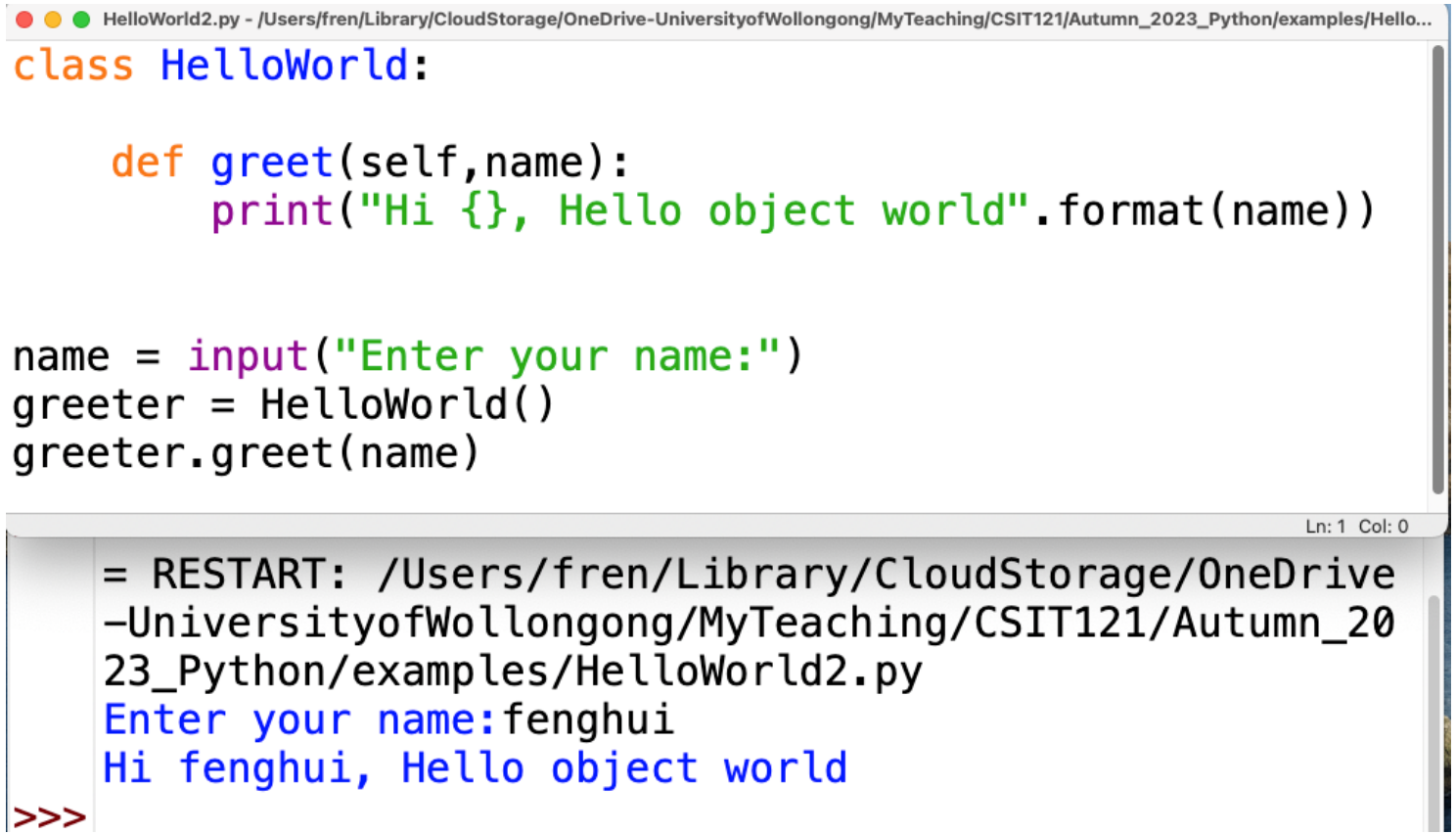
- In Python OOP, a method is a function attached to a class.
- The ‘self’ parameter is a specific instance of that class.
- If you call the method with two different instances (objects), you are calling the same method twice but passing two different objects as the self parameter
- When we call <object>.<method>, we do not have to pass the ‘self’ argument into it.
- Python automatically pass it, i.e., method(object)
- Alternatively, we can invoke the function on the class, and explicitly passing our object as the ‘self’ argument.

‘self’ argument



More arguments

- A method can contain multiple arguments
- Use the dot notation to call the method with all argument values (still no need to include the ‘self’ argument) inside the parentheses



Initialisation of objects

```
1 import math
2
3 class Point:
4     def move(self, x, y):
5         self.x = x
6         self.y = y
7
8     def reset(self):
9         self.move(0, 0)
10
11    def calculate_distance(self, other_point)
12        return math.sqrt(
13            (self.x - other_point.x) ** 2
14            + (self.y - other_point.y) ** 2
15    )
16
17
18 #use it:
19 point1 = Point()
20 point2 = Point()
21 point1.reset()
22 point2.move(5, 0)
23 print(point2.calculate_distance(point1))
24 print(point1.calculate_distance(point1))
```

```
= RESTART: /Users/fren/Library/CloudStora
ge/OneDrive-UniversityofWollongong/MyTeac
hing/CSIT121/Autumn_2023_Python/examples/
Point.py
5.0
0.0
```

What if we forget to call the reset() function?

```
= RESTART: /Users/fren/Library/CloudStora
ge/OneDrive-UniversityofWollongong/MyTeac
hing/CSIT121/Autumn_2023_Python/examples/
Point.py
Traceback (most recent call last):
  File "/Users/fren/Library/CloudStorage/
OneDrive-UniversityofWollongong/MyTeachin
g/CSIT121/Autumn_2023_Python/examples/Poi
nt.py", line 23, in <module>
    print(point2.calculate_distance(point
1))
  File "/Users/fren/Library/CloudStorage/
OneDrive-UniversityofWollongong/MyTeachin
g/CSIT121/Autumn_2023_Python/examples/Poi
nt.py", line 13, in calculate_distance
    (self.x - other_point.x) ** 2
AttributeError: 'Point' object has no att
ribute 'x'
```

Initialisation of objects

- If we try to access an attribute without a value, we will receive an error.
- How do we force users to call ‘reset()’ method every time?
- The better way is to call ‘reset()’ automatically for every new object
- In OOD, a **constructor** refer to a special method that creates and initialises the object when it is created automatically
- The Python initialisation method is the same as any other method but with a special name, **__init__** (the leading and trailing double underscores)

Initialisation of objects

```python
import math

class Point:
    def __init__(self):
        self.x=0
        self.y=0

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

    def reset(self):
        self.move(0, 0)

    def calculate_distance(self, other_point):
        return math.sqrt(
            (self.x - other_point.x) ** 2
            + (self.y - other_point.y) ** 2
    )


#use it:
point1 = Point()
point2 = Point()
#point1.reset()
point2.move(5, 0)
print(point2.calculate_distance(point1))
print(point1.calculate_distance(point1))
```

```python
import math

class Point:
    def __init__(self,x,y):
        self.move(x,y)

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

    def reset(self):
        self.move(0, 0)

    def calculate_distance(self, other_point):
        return math.sqrt(
            (self.x - other_point.x) ** 2
            + (self.y - other_point.y) ** 2
    )


#use it:
point1 = Point(0,0)
point2 = Point(5,0)
#point1.reset()
#point2.move(5, 0)
print(point2.calculate_distance(point1))
print(point1.calculate_distance(point1))
```

```
= RESTART: /Users/fren/Library/CloudStora
ge/OneDrive-UniversityofWollongong/MyTeac
hing/CSIT121/Autumn_2023_Python/examples/
Point.py
5.0
0.0
```

```python
import math

class Point:
    def __init__(self,x=0,y=0):
        self.move(x,y)

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

    def reset(self):
        self.move(0, 0)

    def calculate_distance(self, other_point):
        return math.sqrt(
            (self.x - other_point.x) ** 2
            + (self.y - other_point.y) ** 2
    )


#use it:
point1 = Point()
point2 = Point(5,0)
#point1.reset()
#point2.move(5, 0)
print(point2.calculate_distance(point1))
print(point1.calculate_distance(point1))
```

Explain your class

- It is important to write API documentation to summarize what each object and method does
- The best way to do it is to write it right into your code through the **docstrings**
- Docstrings are simply Python strings enclosed with apostrophes (') or quotation marks (") characters.
- If docstrings are quite long and span multiple lines (the style guide suggests that the line length should not exceed 80 characters), it can be formatted as multi-line strings, enclosed in matching triple apostrophe (''') or triple quote (""") characters.
- A docstring should clearly and concisely summarize the purpose of the class or method it is describing.
- It should explain any parameters whose usage is not immediately obvious, and is also a good place to include short examples of how to use the API.
- Any caveats or problems an unsuspecting user of the API should be aware of should also be noted.

Explain your class

```python
import math

class Point:
    def __init__(self,x=0,y=0):
        """Initialize the position of a new point.
            The x and y coordinates can be specified.
            If they are not, the point defaults to the origin."""
        self.move(x,y)

    def move(self, x, y):
        "Move the point to a new location in 2D space."
        self.x = x
        self.y = y

    def reset(self):
        "Reset the point back to the geometric origin: 0, 0"
        self.move(0, 0)

    def calculate_distance(self, other_point):
        """Calculate the distance from this point to
        a second point using the Pythagorean Theorem"""
        return math.sqrt(
            (self.x - other_point.x) ** 2
            + (self.y - other_point.y) ** 2)

```

```
>>> help(Point)
Help on module Point:

NAME
    Point

CLASSES
    builtins.object
        Point

    class Point(builtins.object)
     |  Point(x=0, y=0)
     |
     |  Methods defined here:
     |
     |  __init__(self, x=0, y=0)
     |      Initialize the position of a new point.
     |                  The x and y
     |      coordinates can be specified.
     |                  If they are not, the
     |      point defaults to
     |                  the origin.
     |
     |  calculate_distance(self, other_point)
     |      Calculate the distance from this point to a
     |              second
```

Class (static) and instance variables

- Instance variables are for data unique to each instance.
- Instance variables are defined inside methods with the prefix ‘self’
- Access via <object>.<InstanceVariable>
- Class (static) variables are for data shared by all instances of the class.
- Class (static) variables are defined outside methods
- Access via cls.<ClassVariable> or <ClassName>.<ClassVariable>

```
5 0
0 0
0 0
Traceback (most recent call last):
  File "/Users/fren/Library/CloudSto
rage/OneDrive-UniversityofWollongong
/MyTeaching/CSIT121/Autumn_2023_Pyth
on/examples/Point.py", line 28, in <
module>
    print(Point.x,Point.y)
AttributeError: type object 'Point'
has no attribute 'x'
```

```python
import math

class Point:

    start_point_x=start_point_y=0 #class variables

    def __init__(self, x=start_point_x, y=start_point_y):
        self.move(x,y)

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

    def reset(self):
        self.move(start_point_x, start_point_y)

    def calculate_distance(self, other_point):
        return math.sqrt(
            (self.x - other_point.x) ** 2
            + (self.y - other_point.y) ** 2)


#use it:
point1 = Point(5,0)
print(point1.x,point1.y)
print(point1.start_point_x,point1.start_point_y)
print(Point.start_point_x,Point.start_point_y)
print(Point.x,Point.y)
```

Class (static) and instance variables

```
1 import math
2
3 class Point:
4
5     start_point_x=start_point_y=0 #class variables
6
7     def __init__(self, x=start_point_x, y=start_point_y):
8         self.move(x,y)
9
10    def move(self, x, y):
11        self.x = x
12        self.y = y
13
14    def reset(self):
15        self.move(start_point_x, start_point_y)
16
17    def calculate_distance(self, other_point):
18        return math.sqrt(
19            (self.x - other_point.x) ** 2
20            + (self.y - other_point.y) ** 2)
21
22    def change_start_point(self,x,y):
23        Point.start_point_x=x
24        Point.start_point_y=y
25
26
27 #use it:
28 point1 = Point(5,0)
29 print(point1.x,point1.y)
30 print(Point.start_point_x,Point.start_point_y)
31 point1.change_start_point(1,1)
32 point2 = Point()
33 print(point2.x,point2.y)
34 print(Point.start_point_x,Point.start_point_y)
```

```
= RESTART: /Users/fren/Library/CloudStorage/OneDrive-U
niversityofWollongong/MyTeaching/CSIT121/Autumn_2023_P
ython/examples/Point.py
5 0
0 0
0 0
1 1
```

- start_point_x and start_point_y are class variables
- Changing the class variable values with Object point1, then we can see the class variable values are modified with Object point2

- However, point2 is not initialised by the new start_point (1,1)
- Can we use Point.start_point_x and Point_start_point_y in the init() method?

Class (static) and instance variables

```
1 import math
2
3 class Point:
4
5     start_point_x=start_point_y=0 #class variables
6
7     def __init__(self, x=Point.start_point_x, y=Point.start_point_y):
8         self.move(x,y)
9
10    def move(self, x, y):
11        self.x = x
12        self.y = y
13
14    def reset(self):
15        self.move(start_point_x, start_point_y)
16
17    def calculate_distance(self, other_point):
18        return math.sqrt(
19            (self.x - other_point.x) ** 2
20            + (self.y - other_point.y) ** 2)
21
22    def change_start_point(self,x,y):
23        Point.start_point_x=x
24        Point.start_point_y=y
25
26
27 #use it:
28 point1 = Point(5,0)
29 print(point1.x,point1.y)
30 print(Point.start_point_x,Point.start_point_y)
31 point1.change_start_point(1,1)
32 point2 = Point()
33 print(point2.x,point2.y)
34 print(Point.start_point_x,Point.start_point_y)
```

```
= RESTART: /Users/fren/Library/CloudStorage/OneDrive-U
niversityofWollongong/MyTeaching/CSIT121/Autumn_2023_P
ython/examples/Point.py
Traceback (most recent call last):
  File "/Users/fren/Library/CloudStorage/OneDrive-Univ
ersityofWollongong/MyTeaching/CSIT121/Autumn_2023_Pyth
on/examples/Point.py", line 3, in <module>
    class Point:
  File "/Users/fren/Library/CloudStorage/OneDrive-Univ
ersityofWollongong/MyTeaching/CSIT121/Autumn_2023_Pyth
on/examples/Point.py", line 7, in Point
    def __init__(self, x=Point.start_point_x, y=Point.
start_point_y):
NameError: name 'Point' is not defined. Did you mean:
'print'?
```

- No, we can not. It is because Point is not created yet in the init() method
- How to resolve this problem?
- We use a sentinel default value instead.

Sentinel Default Value

```python
import math

class Point:

    start_point_x=start_point_y=0 #class variables

    def __init__(self, x=None, y=None):
        if x is None and y is None:
            self.move(Point.start_point_x,Point.start_point_y)
        else:
            self.move(x,y)

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

    def reset(self):
        self.move(start_point_x, start_point_y)

    def calculate_distance(self, other_point):
        return math.sqrt(
            (self.x - other_point.x) ** 2
            + (self.y - other_point.y) ** 2)

    def change_start_point(self,x,y):
        Point.start_point_x=x
        Point.start_point_y=y


#use it:
point1 = Point(5,0)
print(point1.x,point1.y)
print(Point.start_point_x,Point.start_point_y)
point1.change_start_point(1,1)
point2 = Point()
print(point2.x,point2.y)
print(Point.start_point_x,Point.start_point_y)
point3 = Point(3,3)
print(point3.x,point3.y)
print(Point.start_point_x,Point.start_point_y)
```

```
= RESTART: /Users/fren/Library/CloudStorage/OneDrive-U
niversityofWollongong/MyTeaching/CSIT121/Autumn_2023_P
ython/examples/Point.py
5 0
0 0
1 1
1 1
3 3
1 1
```

Class and instance method

- Problem: we still have to create a Point object first, and then we can modify the start point.
- Question: can we modify the start point directly without creating a Point object?
- Answer: yes, we can. We can define a ‘change_start_point()’ method as a class method by marking this method with a ‘**@classmethod**’ decorator.
- Then we can call the method with the class name or the object name, i.e. <ClassName>.<ClassMethod> or <ObjectName>.<ClassMethod>.
- They do the same job but you don’t have to create an object in the first way.

Class and instance method

```python
import math

class Point:

    start_point_x=start_point_y=0 #class variables

    def __init__(self, x=None, y=None):
        if x is None and y is None:
            self.move(Point.start_point_x,Point.start_point_y)
        else:
            self.move(x,y)

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

    def reset(self):
        self.move(start_point_x, start_point_y)

    def calculate_distance(self, other_point):
        return math.sqrt(
            (self.x - other_point.x) ** 2
            + (self.y - other_point.y) ** 2)

    @classmethod
    def change_start_point(cls,x,y):
        cls.start_point_x=x
        Point.start_point_y=y


#use it:
print(Point.start_point_x,Point.start_point_y)
Point.change_start_point(1,1)
print(Point.start_point_x,Point.start_point_y)

point1=Point()
print(point1.move)
print(Point.change_start_point)
```

```
= RESTART: /Users/fren/Library/CloudStorage/OneDrive-U
niversityofWollongong/MyTeaching/CSIT121/Autumn_2023_P
ython/examples/Point.py
0 0
1 1
<bound method Point.move of <__main__.Point object at
0x1048b7a90>>
<bound method Point.change_start_point of <class '__ma
in__.Point'>>
```

- The instance method has access to the object via the self argument. When the method is called, Python replaces the self argument with the instance object.
- Calling class method doesn't have access to the object, but only to the class. Python automatically passes the class as the first argument to the function when we call the class method.
- Summary: Python calls the instance method by passing the object instance as the first argument, and calls the class method by passing the class as the first argument

Static method

- Question: Java has the static method, does Python also have the static method?
- Answer: Yes, Python also has the static method, but it is different from Java's static method
- Java's static method is the same as Python's class method
- Python's static method is the same as Python's normal method, i.e., no object instance or class will be pasted as the first argument.
- Python's static method can be defined by marking this method with a '**@staticmethod**' decorator.
- You are allowed to call a static method via either <objectName>.<StaticMethod>, or <ClassName>.<StaticMethod>

Static method

```python
import math

class Point:

    start_point_x=start_point_y=0 #class variables

    def __init__(self, x=None, y=None):
        if x is None and y is None:
            self.move(Point.start_point_x,Point.start_point_y)
        else:
            self.move(x,y)

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

    def reset(self):
        self.move(start_point_x, start_point_y)

    def calculate_distance(self, other_point):
        return math.sqrt(
            (self.x - other_point.x) ** 2
            + (self.y - other_point.y) ** 2)

    @classmethod
    def change_start_point(cls,x,y):
        cls.start_point_x=x
        Point.start_point_y=y

    @staticmethod
    def greet():
        print("Hello World!")


#use it:
Point.change_start_point(1,1)
print(Point.start_point_x, Point.start_point_y)
Point.greet()
Point.reset()
```

```
= RESTART: /Users/fren/Library/CloudStorage/OneDrive-U
niversityofWollongong/MyTeaching/CSIT121/Autumn_2023_P
ython/examples/Point.py
1 1
Hello World!
Traceback (most recent call last):
  File "/Users/fren/Library/CloudStorage/OneDrive-Univ
ersityofWollongong/MyTeaching/CSIT121/Autumn_2023_Pyth
on/examples/Point.py", line 39, in <module>
    Point.reset()
TypeError: Point.reset() missing 1 required positional
argument: 'self'
```

Python access control

Most OOP languages have a concept of access control.

- (+) public members are accessible from outside the class. The object of the same class is required to invoke a public member.
- (-) private members are accessible from inside the class only.
- (#) protected members are accessible from inside the class or sub-classes

However, Python does not do this!!!

- Python does not believe in enforcing laws that might someday get in your way.
- Technically, all members are publicly available.
- In Python, prefixes are used to interpret the access control of members.
- _ (single underscore) prefixed to a member makes it protected but still can be accessed.
- __ (double underscore) prefixed to a member makes it private. It gives a strong suggestion not to touch it from outside the class. Any attempt to do so will result in an AttributeError.
- Actually, Python performs name mangling of private variables. Every private member with a double underscore can be accessed by adding the prefix '_<ClassName>'. So, it can still be accessed from outside the class, but the practice should be refrained.

Python access control

```
1 import math
2 class Point:
3     start_point_x=start_point_y=0 #class variables
4     __target_point_x=__target_point_y=10
5
6     def __init__(self, x=None, y=None):
7         if x is None and y is None:
8             self.move(Point.start_point_x,Point.start_point_y)
9         else:
10            self.move(x,y)
11
12    def move(self, x, y):
13        self.x = x
14        self.y = y
15
16    def reset(self):
17        self.move(start_point_x, start_point_y)
18
19    def calculate_distance(self, other_point):
20        return math.sqrt(
21            (self.x - other_point.x) ** 2
22            + (self.y - other_point.y) ** 2)
23
24    @classmethod
25    def change_start_point(cls,x,y):
26        cls.start_point_x=x
27        Point.start_point_y=y
28
29    @staticmethod
30    def greet():
31        print("Hello World!")
32
33    def __secrete_message(self):
34        print("This is a secrete!")
35
36 point1=Point()
37 print(point1.__target_point_x, point1.__target_point_y)
38 point1.__secrete_message()
39 print(point1._Point__target_point_x, point1._Point__target_point_y)
40 point1._Point__secrete_message()
```

```
= RESTART: /Users/fren/Library/CloudStorage/OneDrive-U
niversityofWollongong/MyTeaching/CSIT121/Autumn_2023_P
ython/examples/Point.py
Traceback (most recent call last):
  File "/Users/fren/Library/CloudStorage/OneDrive-Univ
ersityofWollongong/MyTeaching/CSIT121/Autumn_2023_Pyth
on/examples/Point.py", line 42, in <module>
    print(point1.__target_point_x, point1.__target_poi
nt_y)
AttributeError: 'Point' object has no attribute '__tar
get_point_x'. Did you mean: 'start_point_x'?

= RESTART: /Users/fren/Library/CloudStorage/OneDrive-U
niversityofWollongong/MyTeaching/CSIT121/Autumn_2023_P
ython/examples/Point.py
10 10
This is a secrete!
```

Python execution control

- A program written in C or Java needs the main() function to indicate the starting point of execution.
- Python does not need the main() function as it is an interpreter-based language.
- The execution of the Python program file starts from the first statement.
- Python uses a special variable called '__name__' (string) to maintain the scope of the code being executed.
- The top-level scope (top-level code executes here) is referred by the value '__main__'
- You can check the scope in Python shell by typing '__name__' in IDEL.

```
>>>
========== RESTART: Shell ==========
>>> __name__
'__main__'
```

Python execution control

- When you execute functions and modules in the interpreter shell directly, they will be executed in the top-level scope '__main__'

```
================ RESTART: Shell ================
>>> __name__
'__main__'
>>> def f1():
...     print(__name__)
...
...
>>> f1()
__main__
```

- We can also save a Python program as a Python file, which may contain multiple functions and statements, such as

addition.py

```
def add(x,y):
    z=x+y
    print('add() executed under the scope: ', __name__)
    return z

x=input('Enter the first number to add: ')
y=input('Enter the secode number to add: ')
result = add(int(x),int(y))
print(x, '+', y,'=', result)
print('Code executed under the scope: ', __name__)
```

Python execution control

- We can use the command prompt/terminal to execute the Python file as a script

```
fren@UW-XY06K990HQ examples % python3 addition.py
Enter the first number to add: 1
Enter the secode number to add: 2
add() executed under the scope:  __main__
1 + 2 = 3
Code executed under the scope: __main__
```

- We can also use the Python file as a module in another file or in interactive shell by importing it.

```
fren@UW-XY06K990HQ examples % Python3
Python 3.11.2 (v3.11.2:878ead1ac1, Feb  7 2023, 10:02:41) [Clang 13.0.0 (clang-1300.0.
29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import addition
Enter the first number to add: 1
Enter the secode number to add: 2
add() executed under the scope:  addition
1 + 2 = 3
Code executed under the scope:  addition
```

Python execution control

- The import statement starts executing from the first statement till the last statement.
- What if we just want to use the ‘add()’ method?
- We can use the special variable ‘__name__’ to check the scope and execute the other statements only when it executes from the command prompt/terminal but not when imported it.

addition.py - /Users/fren/Library/CloudStorage/OneDrive-UniversityofWollongong/MyTeaching/CSIT

```
1 def add(x,y):
2     z=x+y
3     print('add() executed under the scope: ', __name__)
4     return z
5
6 if __name__ == '__main__':
7     x=input('Enter the first number to add: ')
8     y=input('Enter the secode number to add: ')
9     result = add(int(x),int(y))
10
11    print(x, '+', y,'=', result)
12    print('Code executed under the scope: ', __name__)
13
```

```
>>> import addition
>>> addition.add(3,3)
add() executed under the scope:  addition
6
```

Python execution control

- However, when we execute it from the command prompt/terminal, it still executes all statements because of it being executed in the top-level scope '__main__'.
- Thus, value of the '__name__' allows the Python interpreter to determine whether a module is intended to be an executable script.

addition.py - /Users/fren/Library/CloudStorage/OneDrive-UniversityofWollongong/MyTeaching/CSIT

```
1 def add(x,y):
2     z=x+y
3     print('add() executed under the scope: ', __name__)
4     return z
5
6 if __name__ == '__main__':
7     x=input('Enter the first number to add: ')
8     y=input('Enter the secode number to add: ')
9     result = add(int(x),int(y))
10
11    print(x, '+', y,'=', result)
12    print('Code executed under the scope: ', __name__)
13
```

```
fren@UW-XY06K990HQ examples % Python3 addition.py
Enter the first number to add: 1
Enter the secode number to add: 2
add() executed under the scope:  __main__
1 + 2 = 3
Code executed under the scope:  __main__
```

A case study

Design and implement a Python program with OOD & OOP to calculate the length of a given line segment.

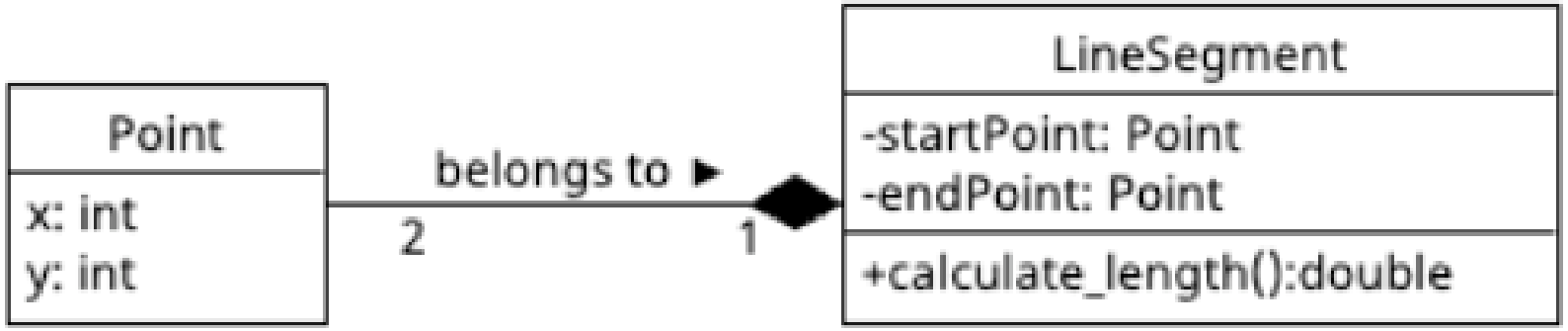
Step 1: OOA

- To calculate the length of a given line segment, the LineSegment class can be defined. The LineSegment class should contain two points, i.e., the start point and the end point. The length of the line segment is the Cartesian distance between the two points.
- So the LineSegment class should have two points (attribute) and can calculate the Cartesian distance of two points (behavior)
- To represent a point, the Point class can be defined. The Point class should contain the coordinate (x, y) of the point (attribute).

A case study

Design and implement a Python program with OOD & OOP to calculate the length of a given line segment.

Step 2: OOD (UML)



A case study

Design and implement a Python program with OOD & OOP to calculate the length of a given line segment.

Step 3: OOP (Python)

Step 4: use the '__name__' variable to ensure your testing cases are only executed in the command prompt/terminal.

A further question: can you modify your design and code to calculate the total length of consecutive line segments?

```python
import math

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

class LineSegment:
    def __init__(self, start_point, end_point):
        self.start_point = start_point
        self.end_point = end_point

    def calculate_length(self):
        dx = self.end_point.x - self.start_point.x
        dy = self.end_point.y - self.start_point.y
        length = math.sqrt(dx**2 + dy**2)
        return length


# Example usage:
if __name__ == '__main__':
    start = Point(0, 0)
    end = Point(3, 4)
    line = LineSegment(start, end)
    length = line.calculate_length()
    print("Length of the line segment:", length)
```

```
= RESTART: /Users/fren/Library/CloudStorage/OneDrive-U
niversityofWollongong/MyTeaching/CSIT121/Autumn_2023_P
ython/examples/LineSegment.py
Length of the line segment: 5.0
```

Suggested reading

Python 3 Object-Oriented Programming

- Preface
- Chapter 2: Objects in Python

Python

- https://www.python.org/