

Week 2:

Arnamoy Bhattacharyya

*Most slides inspired by Bogdan Simion and Danny Heap

Attributes in Python

Ability or characteristics which something or someone has

Binding attributes to objects – general concept in Python

Class Attribute vs Instance Attribute

```
>>> class A:
...     a = "I am a class attribute!"
...
>>> x = A()
>>> y = A()
>>> x.a = "This creates a new instance attribute for x!"
>>> y.a
'I am a class attribute!'
>>> A.a
'I am a class attribute!'
>>> A.a = "This is changing the class attribute 'a'!"
>>> A.a
"This is changing the class attribute 'a'!"
>>> y.a
"This is changing the class attribute 'a'!"
>>> # but x.a is still the previously created instance variable:
...
>>> x.a
'This creates a new instance attribute for x!'
>>>
```

Class Robot: Class Level Attribute

```
class Robot:

    Three_Laws = (
        """A robot may not injure a human being or, through inaction, allow a human being to come to harm.""",
        """A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law., """,
        """A robot must protect its own existence as long as such protection does not conflict with the First or Second Law."""
    )

    def __init__(self, name, build_year):
        self.name = name
        self.build_year = build_year

    # other methods as usual
```

```
>>> from robot_asimov import Robot
>>> for number, text in enumerate(Robot.Three_Laws):
...     print(str(number+1) + ":\n" + text)
...
1:
A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2:
A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.,
3:
A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.
>>>
```

Exercise: Class Level Attribute

Create a class *Counter* that will have a “counter” incremented whenever we create an object of that class and decremented whenever we delete an instance of that class:

Hint: When an instance is deleted like:

```
c1 = counter()
```

```
del c1
```

The `__del__` magic method is called

Class Attributes and Methods

Ability or characteristics which something or someone has

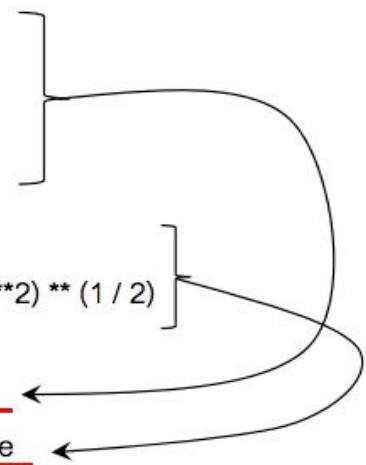
Binding attributes to objects – general concept in Python

Method differs from function in 2 aspects:

- Belongs to a class and it is defined within a class
- First parameter has to be a reference to the instance which called the method – “self”

Wrong way of designing a Point class

```
>>> class Point:
...     pass
...
>>> def initialize(point, x, y):
...     point.x = x
...     point.y = y
...
>>> def distance(point):
...     return (point.x**2 + point.y**2) ** (1 / 2)
...
>>> Point.__init__ = initialize
>>> Point.distance = distance
>>> p2 = Point(12, 5)
>>> p2.distance()
13.0
```



Class design: More examples

Rational fractions

- Although Python has a built-in type for floating-point numbers, there is no built-in type for representing rational numbers
- Similarly, we want to design and implement a class for rational numbers.
- As before, we follow the design recipe for classes

Rational Numbers

- Specification:

Rational numbers are ratios of two integers p/q , where p is called the numerator and q is called the denominator. The denominator q is non-zero.

Operations on rationals include addition, multiplication, and comparisons: $>$, $<$, $>=$, $<=$, $=$

Designing the Rational Class

Step 1: Read the Specs

Rational numbers are ratios of two integers p/q , where p is called the numerator and q is called the denominator. The denominator q is non-zero.

Operations on *rationals* include *addition*, *multiplication*, and *comparisons*: $=$, $<>$, $<$, $>$, $<=$, $>=$

Designing the Rational Class

- Step 2 – define a class API:
 - 1. Choose a **class name** and write a **brief description** in the class docstring
 - 2. Write some **examples** of client code that uses your class
 - Put this code in the “main block”
 - 3. Decide what operations your class should provide as public **methods**, for each method declare an **API** (examples, type contract, header, description)
 - Refer to Function design recipe
 - 4. Decide which **attributes** your class should provide without calling a method, list them in the class docstring

Step 3 - implement the class:

- 1. body of special methods Always: **`__init__`**, **`__eq__`**, and **`__str__`**, Others (as needed): **`__add__`**, **`__mul__`**, **`__lt__`**, etc.
- 2. body of other methods

Python recognizes the names of special methods such as `__init__`, `__eq__`, `__str__`, `__add__`, and `__mul__` and has shortcuts (aliases) for them.

- This syntactic sugar doesn't change the semantics (meaning) of these methods, but may allow more manageable code.

Special methods

- Rational

Rational numbers are ratios of two integers p/q , where p is called the numerator and q is called the denominator. The denominator q is non-zero.

Operations on *rationals* include *addition*, *multiplication*, and *comparisons*: $=$, $<>$, $<$, $>$, $<=$, $>=$

Attributes for Rational

Special Attributes in python (magic methods)

`== -__eq__`

`> __gt__`

`< __lt__`

`print(object) __str__`

Created automatically with empty body

You can implement your corresponding code

Protecting against mistakes

Bad inputs can cause programs to crash

For Rational Class:

- What if num and denom are not integers?
- What if denom is 0?

Next session

- Managed attributes, properties
- Types within types ... composition!
- Generalize classes with inheritance