Week 7: Objec Oriented Programming (OOP)

Computational Tools and Techniques in STEM

Mar 12-14, 2019

Learning Goals

- **L1**: Instance, static and class methods
- **L2**: Composition
- **L3**: Python special methods
- **L4**: Principles of OOP

Specials/Dunders/Magics

The table below provides an overview of the most important special methods.

Construction	Meaning2
`a.__init__(self, args)`	constructor: `a = A(args)`
`a.__del__(self)`	destructor: `del a`
`a.__call__(self, args)`	call as function: `a(args)`
`a.__str__(self)`	pretty print: `print a, str(a)`
`a.__repr__(self)`	representation: `a = eval(repr(a))`
`a.__add__(self, b)`	`a + b`
`a.__sub__(self, b)`	`a - b`
`a.__mul__(self, b)`	`a*b`
`a.__div__(self, b)`	`a/b`
`a.__radd__(self, b)`	`b + a`
`a.__rsub__(self, b)`	`b - a`
`a.__rmul__(self, b)`	`b*a`
`a.__rdiv__(self, b)`	`b/a`
`a.__pow__(self, p)`	`a**p`
`a.__lt__(self, b)`	`a < b`
`a.__gt__(self, b)`	`a > b`
`a.__le__(self, b)`	`a <= b`
`a.__ge__(self, b)`	`a => b`
`a.__eq__(self, b)`	`a == b`
`a.__ne__(self, b)`	`a != b`
`a.__bool__(self)`	boolean expression, as in `if a:`
`a.__len__(self)`	length of a (int): `len(a)`
`a.__abs__(self)`	`abs(a)`

Figure: Picture source: `http://hplgit.github.io/primer.html/doc/pub/class/._class-solarized007.html`

Instance, static and class methods

```
class DemoMethods:
"""This class contains all three different types of methods
                                                    . """
def instance_method(self):
    # needs object
    return 'I am an instance_method', self

# @classmethod is called a decorator in Python
@classmethod
def class_method(cls):
    # needs class
    return 'I am a class method', cls

# @staticmethod is also a decorator
@staticmethod
def static_method():
    # does not need class or object
    return 'I am a static method'
```

__str__ vs __repr__

```
# For call to repr(). Prints more accurate information
    def __repr__(self):
        return 'Particles(%s, %s)' % (self.radius, self.shape)

    # For call to str(). Prints information in nicer format
    def __str__(self):
        return '%s + i%s' % (self.radius, self.shape)
```

Composition

Can you “compose” two class objects?

Yes! You can write a method in the class definition that does it.

When composing two different objects using a method, how do you refer to them?

One instance is conventionally referred to as “self”, while the other one is usually referred to as “other” in class defintion.

Principles of OOP

Inheritance: Changes you make to your project.

Encapsulation: Protecting the data.

Abstraction: Hiding the complexity.

Polymorphism: To use common interface for multiple form.

Additional concepts:

Overriding: A method in subclass having a different implemention of a method defined in the superclass.

Overloading: Different method implementations that have the same name and functionality, but take different types of arguments. They maybe present in the same class.

Project

- Implement assigning of virtual cells to particles in your code.
- Tag the particles in your code.