Basic Python

ML4HEP 2026 Pre-school Lectures

Subir Sarkar, SINP

Lecture 4, 12/05/2026

Course Outline

- Object Oriented Programming

Object Oriented Programming - I

Object Oriented Programming

```
# account.py
class BankAccount:
  def __init__(self, amount=0): # constructor
    self.__balance = amount
  def withdraw(self, amount): # methods
    self.__balance -= amount
  def deposit(self, amount):
    self.__balance += amount
  def balance(self):
    return self.__balance
```

Object Oriented Programming

```
>>> from account import BankAccount
>>> a = BankAccount(1000) # instances
>>> b = BankAccount()
>>> a.deposit(100)  # -> deposit(a, 100)
>>> b.deposit(50)
>>> b.withdraw(10)
>>> a.withdraw(10)
```

The *self* parameter

- The first argument of every class instance method, including `__init__` and `__del__`, is an explicit reference to the current instance of the class
- by convention, this argument is always named *self, but it is not a keyword*

```
def __init__(self, amount=0)
```

- In the `__init__` method, *`self`* refers to the object being created
- In other methods, *`self`* refers to the object
  - whenever an object calls its method, the object itself is passed as the first argument
- The constructor is defined with two arguments, while an object is created passing none!

```
a = BankAccount()  # 2nd parameter has default value
```

Operator Overloading

```
class Integer:
  def __init__(self, v=0):
    self._value = v
  def set(self, v):
    self._value = v
  def __add__(self, value):
    return self._value + value
  def __sub__(self, value):
    return self._value - value
  def __mul__(self, value):
    return self._value * value
  def __truediv__(self, value):
    return self._value / value
  def __lt__(self, value):
    return self._value < value
```

Operator Overloading

```
class Integer:
  def __init__(self, v=0):
    self._value = v
  def set(self, v):
    self._value = v
  def __add__(self, value):
    return self._value + value
  def __sub__(self, value):
    return self._value - value
  def __mul__(self, value):
    return self._value * value
  def __truediv__(self, value):
    return self._value / value
  def __lt__(self, value):
    return self._value < value
```

```
if __name__ == '__main__':
  d = Integer(10)
  print(type(d))

  print(d + 2)
  print(d - 2)
  print(d * 2)
  print(d / 5)

  if d < 5:
     print("Less than 5")
  else:
     print("Not less than 5")

  d = 10  # d.set(10)
  if d == 5:
    print("Yes!")
  else:
    print("no...")
```

Operator Overloading

Operator	Expression	Internally
Addition	p1 + p2	p1.__add__(p2)
Subtraction	p1 - p2	p1.__sub__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Remainder (modulo)	p1 % p2	p1.__mod__(p2)
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise OR	p1 | p2	p1.__or__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)
Bitwise NOT	~p1	p1.__invert__()

Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)

Property

```
class Track:
  def __init__(self, artist, title, duration):
    self.__artist = artist
    self.__title = title
    self.__duration = duration

  @property
  def artist(self): # artist instead of artist()
    return self.__artist

  @property
  def title(self):
    return self.__title

  @property
  def duration(self):
    return self.__duration

  def __str__(self):
    return '%s - %s - %s' % (self.__artist, self.__title, self.__duration)
```

Property

```
if __name__ == "__main__":
    track = Track("Ravi Shankar", "Bairagi Todi", 25)
    print(track)

    # access directly
    print ('%s - %s - %s' %
      (track.artist, track.title, track.duration))
```

Object Oriented Programming - II

Class, Subclass

```
class Person:  # base class
  def __init__(self, name, age): # constructor
    self.__name = name
    self.__age = age

  def name(self):
    return self.__name

  def age(self):
    return self.__age

  def introduce(self):              # method
    return 'Person - name: %s, age: %s'% (self.name(), self.age())

  def __str__(self):
    return 'Person - name: %s, age: %s'% (self.name(), self.age())
```

Class, Subclass

```
class Student(Person): # derived class
  """ A student class
  """
  def __init__(self, name, age, id):
    Person.__init__(self, name, age)
    self.__id = id
  def introduce(self):  # over-ridden method
    return 'Student - name: %s,age: %s,id: %d'% \
           (self.name(), self.age(), self.__id)
```

Class, Subclass

```
if __name__ == "__main__":
    p1 = Person('Tendulkar', 50)
    p2 = Person('Dravid', 50)
    p2.nickname = 'The Wall' #What's that?

    print(p2.introduce(), \
      'nickname: ', p2.nickname)

    s1 = Student('Rahane', 32, 100)
    print(s1.introduce())
```

Composition & Inheritance

```
# composition
class Stack:
  def __init__(self):
    self._stack = []

  def push(self,object):
    self._stack.append(object)

  def pop(self):
    return self._stack.pop()

  def length(self):
    return len(self._stack)

  def __repr__(self):
    return repr(self._stack)
```

Composition & Inheritance

```
# composition
class Stack:
  def __init__(self):
    self._stack = []

  def push(self, object):
    self._stack.append(object)

  def pop(self):
    return self._stack.pop()

  def length(self):
    return len(self._stack)

  def __repr__(self):
    return repr(self._stack)
```

```
# inheritance
class StackD(list):
  def push(self, obj):
    self.append(obj)
```

Composition & Inheritance

```
if __name__ == "__main__":
    # composition
    s = Stack()
    s.push("Multiverse")
    s.push(42)
    s.push([3,4,5])
    print(s)
    x = s.pop()
    y = s.pop()
    print(s)
```

Composition & Inheritance

```
if __name__ == "__main__":
    # composition
    s = Stack()
    s.push("Multiverse")
    s.push(42)
    s.push([3,4,5])
    print(s)
    x = s.pop()
    y = s.pop()
    print(s)
```

```
# inheritance
s = StackD() # etc.
```

class & static methods

```
from datetime import date

class Person:
  def __init__(self, name, age):
      self.name = name
      self.age = age

  @classmethod
  def fromBirthYear(classname, name, year):
    return classname(name, date.today().year -year)

  @staticmethod
  def isAdult(age):
      return age > 18
```

class & static methods

```
>>> p0 = Person()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __init__() takes exactly 3 arguments (1
given)

>>> p1 = Person('Ajinka', 21)
>>> p2 = Person.fromBirthYear('Ajinka', 1996)
>>> print p1.age
>>> print p2.age # print the result
>>> print Person.isAdult(22)
```