Python for Data Science

Pre-school Lecture Series
(HEPAP-DAS, EHEP2024)

November 3, 2023 - Lecture 5

Subir Sarkar, SINP
subir.sarkar@cern.ch

Object Oriented Programming

Object Oriented Programming

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

Object Oriented Programming

```
>>> from account import BankAccount
>>> a = BankAccount(1000) # instances
>>> b = BankAccount()
>>> a.deposit(100) # -> deposit(a, 100)
100
>>> b.deposit(50)
50
>>> b.withdraw(10)
40
>>> a.withdraw(10)
90
```

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, amnt=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 __gt__(self, value):
    if self._value > value:
      return 1
    else:
      return 0
  def __lt__(self, value):
    if self._value < value:
      return 1
    else:
      return 0
  def __eq__(self, value):
    if self._value == value:
      return 1
    else:
      return 0
```

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 __gt__(self, value):
    if self._value > value:
      return 1
    else:
      return 0
  def __lt__(self, value):
    if self._value < value:
      return 1
    else:
      return 0
  def __eq__(self, value):
    if self._value == value:
      return 1
    else:
      return 0
```

```
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))
```

Class, Subclass

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

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


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'

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

    s1 = Student('Rahane', 32)
    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
    # We can use StackD the same way
    s = StackD() # etc.
```

`__str__, __repr__, __call__`

__repr__

```
class Rectangle:
  def __init__(self, w=10, h=10):
    self._width  = w
    self._height = h
  def __repr__(self):
    return 'Rectangle (width=%d,height=%d)' \
    % (self._width, self._height)
```

__repr__

```
class Rectangle:
  def __init__(self, w=10, h=10):
    self._width  = w
    self._height = h
  def __repr__(self):
    return 'Rectangle (width=%d,height=%d)' \
    % (self._width, self._height)

if __name__ == "__main__":
  r = Rectangle()
  print(r)
  print(str(r))

  r1 = Rectangle(2,2)
  print(r1)
```

__str__

```
class Rectangle:
  def __init__(self, w, h):
    self._width = w
    self._height = h

  def area(self):
    return self._width * self._height
```

__str__

```
class Rectangle:
  def __init__(self, w, h):
    self._width = w
    self._height = h

  def area(self):
    return self._width * self._height


if __name__ == '__main__':
  r1 = Rectangle(100,20)
  print(r1)
<__main__.Rectangle object at 0x7f93a8706fd0>
```

str

```
class Rectangle:
  def __init__(self, w, h):
    self._width = w
    self._height = h

  def area(self):
    return self._width * self._height

  def __str__(self):
    return '(Rectangle: %s, %s)' % \
    (self._width, self._height)
```

__str__

```
class Rectangle:
  def __init__(self, w, h):
    self._width = w
    self._height = h

  def area(self):
    return self._width * self._height

  def __str__(self):
    return '(Rectangle: %s, %s)' % \
    (self._width, self._height)

if __name__ == '__main__':
  r1 = Rectangle(100,20)
  print(r1)
(Rectangle: 100, 20)
```

__call__

```
class Foo:
   def __call__(self):
      print('called')

foo_instance = Foo()

# calls the __call__ method
foo_instance()
```

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', 27)
>>> p2 = Person.fromBirthYear('Ajinka',
1996)
>>> print p1.age
>>> print p2.age # print the result
>>> print Person.isAdult(22)
```

Built-in class attributes

```
>>> class Account():
...     ''' A class defines a new data type '''
...     pass
...
>>> dir(Account)
['__class__', '__delattr__', '__dict__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__']
>>> type(Account)
<class 'type'>
>>> type(Account())
<class '__main__.Account'>
```

Built-in class attributes

```
>>> Account.__name__
'Account'
>>> Account.__doc__
'A class defines a new data type'
>>> Account.__module__
'__main__'
>>> Account.__bases__
(<class 'object'>,)
```

Derived class attributes

```
>>> dir(Student)
['__doc__', '__init__', '__module__']
>>> Student.__bases__
(__main__.Person,)
>>> Student.__doc__
' A student class\n  '
```

Introspection

Introspection in Computing

- Ability to determine object type, find attributes and capabilities at runtime
- Support for introspection is an integral part of Python
  - runs deep and wide throughout the language
- Other languages
  - C++ has minimal support (RTTI)
    - External libraries provide necessary support (reflex)
  - Java has extensive support (Reflection)
  - Perl: adequate support, not used extensively
  - Ruby, Julia: same level as Python

Introspection – Python Tools

- `dir()`
- `type(), id(), str(), repr()`
- `callable()`
- `hasattr(), getattr(), setattr(), delattr()`
- `isinstance(), issubclass()`
- `modules: types, inspect`
- Reference:
  - zetcode.com/lang/python/introspection
  - www.ibm.com/developerworks/linux/library/l-pyint/index.html
  - www.diveintopython.net/power_of_introspection/

Object attributes

- access and manipulate object attributes with
  - `getattr(obj, name[, default])`
  - `hasattr(obj, name)`
  - `setattr(obj, name, value)`
  - `delattr(obj, name)`

type()

```
>>> type(dir)
>>> type(dir())

>>> import sys
>>> import math

>>> type(len)              # function
>>> type(sys)              # module
>>> type(type)             # type

>>> type(1)
<class 'int'>
>>> type(1) is int
True
>>> type(int) is type
True
```

str(), repr()

```
>>> import datetime
>>> dir(datetime)
['MAXYEAR', 'MINYEAR', '__doc__', '__file__',
'__name__', '__package__', 'date', 'datetime',
'datetime_CAPI', 'time', 'timedelta', 'tzinfo']

>>> type(datetime.datetime)
<class 'type'>
>>> today = datetime.datetime.now()
>>> type(today)
<class 'datetime.datetime'>

>>> today
datetime.datetime(2014, 4, 14, 19, 58, 35, 828815)
>>> str(today)
'2014-04-14 19:58:35.828815'
>>> repr(today)
'datetime.datetime(2014, 4, 14, 19, 58, 35, 828815)'
```

Class, Subclass

```
class Person(object):
  def __init__(self, name, age): # constructor
    self.name = name
    self.age = age
  def intro(self):                    # method
    return 'name:%s and age:%s'%(self.name,self.age)

class Student(Person):
  pass

p1 = Person('Sachin', 40)
p2 = Person('Rahul', 40)
p2.nickname = 'The Wall'

s1 = Student('Ajinka', 21)
```

isinstance(), issubclass()

```
print type(Person)
print type(p1)

print type(Student)
print type(s1)

print isinstance(p1, Person)
print isinstance(s1, Person)
print isinstance(p1, Student)
print isinstance(s1, Student)

print issubclass(Person, Student)
print issubclass(Student, Person)
```

hasattr(), getattr()

```
>>> li = ['Larry', 'Guido', 'Yukihiro']
>>> li.pop
<built-in method pop of list object at 0x7fe80e6293b0>
>>> getattr(li, 'pop')
<built-in method pop of list object at 0x7fe80e6293b0>

>>> f = getattr(li, 'pop')
>>> f()
'Yukihiro'
>>> getattr(li, 'pop')()
'Guido'

>>> getattr(li, 'append')('Brent')
>>> li
['Larry', 'Brent']
>>> hasattr(li, 'clear')
False
>>> hasattr(li, 'insert')
True
```

callable()

```
>>> import string
>>> type(string)
<class 'module'>

>>> dir(string)
[..., 'find', 'hexdigits', 'index', 'index_error', 'join',
'joinfields', 'letters', 'ljust', 'lower', 'lowercase', 'lstrip',
'maketrans', 'octdigits', 'printable', 'punctuation', 'replace',
'rfind', ... 'upper', 'uppercase', 'whitespace', 'zfill']

>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

>>> string.join
<function join at 0x7f19f2bc42a8>

>>> callable(string.join)
True

>>> callable(string.punctuation)
False
>>>
```