Объектно-ориентированный Python и биоинформатика

Махортов Сергей Дмитриевич
ВГУ, ФКН, кафедра ПиИТ
http://www.cs.vsu.ru/msd
Email: msd_exp@outlook.com

Источники

- *Дауни Аллен*. Основы Python. Научитесь думать как программист / Аллен Б. Дауни ; пер. с англ. – М.: Манн, Иванов и Фербер, 2021. – 304 с.
- *Гэддис Т*. Начинаем программировать на Python. – 5-е изд.: Пер. с англ. - СПб.: БХВ-Петербург, 2022. – 880 с.
- *David A. Hendrix*. Applied Bioinformatics, © Oregon State University (2019), 117 p.
- *Katja Schuerer and Catherine Letondal*. Python course in Bioinformatics, © Pasteur Institute (2004), 178 p.

Пользовательские типы данных

- В Python заданный программистом тип называется *классом* (*class*)
- Данные, относящиеся к классу, называются эго *экземплярами*
- Пример: класс *Point* – точка в двумерном пространстве

```
>>> # Представление точки в двумерном пространстве
... class Point:
...     pass
...
>>> print(Point)
<class '__main__.Point'>
>>> blank = Point()
>>> print(blank)
<__main__.Point object at 0x000002955D80A550>
```

Атрибуты класса

- Элементам экземпляра класса можно присваивать значения, используя точечную нотацию

```
>>> blank.x = 3.0
>>> blank.y = 4.0
```

- Значение присваиваются именованным элементам объекта, которые называются *атрибутами* (*attributes*)

```
>>> print(blank.y)
4.0
>>> x = blank.x
>>> print(x)
3.0
```

Функция с аргументом-объектом

```
>>> def printPoint(p):
...     print('(%g, %g)' % (p.x, p.y))
...
>>>
```

- Функция *printPoint*() в качестве аргумента принимает точку (экземпляр класса) и отображать ее в привычной математической форме
- В качестве аргумента ей можно передать объект blank

```
>>> printPoint(blank)
(3, 4)
```

Изменение объекта

- Экземпляры классов в Python являются *изменяемыми* объектами
- Поэтому параметр *p* – это *псевдоним* объекта *blank*

```
>>> def pointInc(p):
...     p.x += 1
...     p.y += 1
...
>>> pointInc(blank)
>>> print(blank.x)
4.0
>>> print(blank.y)
5.0
```

Прямоугольники

```
# Определение прямоугольника. Атрибуты: width, height, corner.
class Rectangle:
  pass
box = Rectangle()
box.width = 100.0
box.height = 200.0
box.corner = Point()
box.corner.x = 0.0
box.corner.y = 0.0
```

- *box.corner.x*: «Обратиться к объекту *box*, выбрать атрибут *corner*, затем обратиться к этому объекту и выбрать его атрибут *x*»
- Объект, который является атрибутом другого объекта, называется *вложенным* (*embedded*)

Возвращение экземпляров функциями

- Функции могут возвращать экземпляры классов

```
def findCenter(rect):
  p = Point()
  p.x = rect.corner.x + rect.width / 2
  p.y = rect.corner.y + rect.height / 2
  return p
```

- В качестве аргумента передается *box*, результирующая точка присваивается переменной *center*

```
>>> center = findCenter(box)
>>> printPoint(center)
(50, 100)
```

Объекты изменяемы – присваивания

- Объект можно изменять, присваивая значения его атрибутам

```
box.width += 50
box.height += 100
```

- В момент присваивания объекта создается его псевдоним (не копия!). Поэтому при изменении объекта меняется и объект-псевдоним.

```
>>> box1 = box
>>> print(box1.width)
150.0
>>> box1.width += 50
>>> print(box.width)
200.0
```

Объекты изменяемы – аргумент функции

- Возможно также определение функций, изменяющих объекты

```
def growRectangle(rect, dwidth, dheight):
   rect.width += dwidth
   rect.height += dheight
```

- При использовании функции получается следующий результат

```
>>> print(box.width, ', ', box.height)
200.0 , 300.0
>>> growRectangle(box, 50, 100)
>>> print(box.width, ', ', box.height)
250.0 , 400.0
```

- Параметр *rect* – это псевдоним для *box*, поэтому после вызова функции *box* тоже меняется

Копирование объектов

- Альтернатива псевдонимам – копирование («клонирование») объектов

```
p1 = Point()
p1.x, p1.y = 3.0, 4.0
import copy
p2 = copy.copy(p1)
```

- p1 и p2 содержат одинаковые (но не общие) данные, и они не являются одним и тем же объектом (псевдонимами) класса

```
>>> printPoint(p1)
(3, 4)
>>> printPoint(p2)
(3, 4)
>>> print(p1 is p2)
False
>>> print(p1 == p2)
False
```

Копирование сложных объектов

- Применение copy() для дублирования прямоугольника скопирует объект *Rectangle*, но не вложенный объект *Point*

```
>>> box2 = copy.copy(box)
>>> print(box2 is box)
False
>>> print(box2.corner is box.corner)
True
```

- Такая операция называется *поверхностным копированием* (*shallow copy*),
- Она копирует объект и все содержащиеся в нем ссылки, но не вложенные объекты
- Для большинства приложений это не тот результат, который требуется получить

«Глубокое» копирование объектов

- Модуль *copy* предоставляет метод *deepcopy*()
- Он копирует не только указанный объект, но и все его «внутренние» объекты, при любом уровне вложенности

```
>>> box3 = copy.deepcopy(box)
>>> print(box3 is box)
False
>>> print(box3.corner is box.corner)
False
```

- Теперь *box3* и *box* – совершенно разные объекты

Отладка объектов

- При работе с объектами можно столкнуться со специфическими ошибками (*исключениями*)

```
>>> p = Point()
>>> p.x = 3
>>> p.y = 4
>>> print(p.z)
AttributeError: 'Point' object has no attribute 'z'
```

- Можно уточнить, к какому типу принадлежит объект

```
>>> print(type(p))
<class '__main__.Point'>
```

- Проверка, является ли объект экземпляром конкретного класса

```
>>> print(isinstance(p, Point))
True
```

Отладка объектов - продолжение

- Проверка наличия атрибута

```
>>> print(hasattr(p, 'x'))
True
>>> print(hasattr(p, 'z'))
False
```

- Использование аппарата *исключений* (*exceptions*)

```
>>> try:
...     x = p.x
... except AttributeError:
...     x = 0
>>> print(x)
3
```

- Эти средства упрощают написание программ со свойством *полиморфизма* типов (будет рассмотрено позднее)

Словарь терминов

- Класс. Тип, определяемый программистом. Класс может использоваться для создания своих экземпляров («представителей»).
- Экземпляр. Объект, который принадлежит классу.
- Создание экземпляра. Создание нового объекта указанного класса.
- Атрибут. Одно из именованных значений, содержащихся в объекте.
- Вложенный объект. Объект, являющийся атрибутом другого объекта.
- Поверхностная копия. Копия содержимого объекта, включая все ссылки на вложенные объекты. Реализуется функцией *copy*() модуля *copy*.
- Глубокая копия. Полная копия содержимого объекта, в том числе объектов любого уровня вложенности. Реализуется функцией *deepcopy*() модуля *copy*.