介绍

张晓平

1 Python 介绍

Python 是一种高级的、动态的多范型编程语言。很多时候，大家会说 Python 看起来简直和伪代码一样，因为你可以使用很少几行可读性很高的代码来表达很有力的想法。举个例子，下面是经典快速排序算法的 Python 实现：

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)

print(quicksort([3,6,8,10,1,2,1]))
# Prints "[1, 1, 2, 3, 6, 8, 10]"
```

1.1 Python versions

目前 Python 有两个支持的版本，分别是 2.7 和 3.4。这有点让人迷惑，Python 3.0 引入了很多不可向下兼容的变化，2.7 下的代码有时候在 3.4 下是行不通的。

如何查看版本呢？可使用如下命令：

```
python --version
```

1.2 Python、numpy、scipy、matplotlib 与 Jupyter notebook 安装

Python 安装及教程请查看廖雪峰的官方网站。

关于 numpy、scipy、matplotlib 与 Jupyter notebook 的安装请查看：

- Ubuntu 下 Jupyter Notebook 的安装与使用
- Windows, Ubuntu 下 Numpy, Scipy, matplotlib, jupyter notebook 安装配置

1.3 基本数据类型

和大多数编程语言一样，Python 拥有一系列的基本数据类型，比如整型、浮点型、布尔型和字符串等。这些数据类型的使用方式和在其他语言中的使用方式是类似的。

1.3.1 Numbers

整型和浮点型的使用和其他语言类似。

```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

注意：Python 没有类似于 ++ 和 -- 的一元运算的操作。Python 也有内置的长整型和复数类型，具体细节可查看文档。

1.3.2 Booleans

Python 实现了所有的布尔逻辑，但用的是英语，而不是我们习惯的操作符 (如&&, || 等):

```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

名称	操作符
小于	<
大于	>
小于等于	<=
小于等于	>=
等于	==
不等于	!=
逻辑与	and
逻辑或	or
逻辑非	not

1.3.3 Strings

Python 对字符串的支持非常强大。

```
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print(hello)       # Prints "hello"
print(len(hello))  # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)  # prints "hello world 12"
```

字符串对象有一系列常用的方法，例如：

```
s = "hello"
print(s.capitalize())  # Capitalize a string; prints "Hello"
print(s.upper())       # Convert a string to uppercase; prints "HELLO"
print(s.rjust(7))      # Right-justify a string, padding with spaces; prints "
hello"
print(s.center(7))     # Center a string, padding with spaces; prints " hello "
print(s.replace('l', '(ell)'))  # Replace all instances of one substring with
another;
                               # prints "he(ell)(ell)o"
print('  world '.strip())  # Strip leading and trailing whitespace; prints "world"
```

如果想详细查看字符串方法，请查看文档。

1.4 容器（Containers）

Python 包含了几个内置的容器类型：列表（lists）、字典（dictionaries）、集合（sets）、元组（tuples）

1.4.1 列表（list）

列表就是 Python 中的数组，但列表长度可变，且能包含不同类型的元素。

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])  # Prints "[3, 1, 2] 2"
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
xs[2] = 'foo'     # Lists can contain elements of different types
print(xs)         # Prints "[3, 1, 'foo']"
xs.append('bar')  # Add a new element to the end of the list
print(xs)         # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()      # Remove and return the last element of the list
print(x, xs)      # Prints "bar [3, 1, 'foo']"
```

列表的细节，可以查阅文档。

切片（Slicing）：为了一次性地获取列表中的多个元素，Python 提供了一种简洁的语法，这就是切片。

```
nums = list(range(5))     # range is a built-in function that creates a list of
integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])          # Get a slice from index 2 to 4 (exclusive); prints "[2,
3]"
print(nums[2:])           # Get a slice from index 2 to the end; prints "[2, 3, 4]"
```

```
print(nums[:2])             # Get a slice from the start to index 2 (exclusive);
prints "[0, 1]"
print(nums[:])              # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])            # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]          # Assign a new sublist to a slice
print(nums)                 # Prints "[0, 1, 8, 9, 4]"
```

在 Numpy 数组的内容中，我们会再次看到切片语法。

循环（Loops）：我们可以这样遍历列表中的每一个元素：

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
# Prints "cat", "dog", "monkey", each on its own line.
```

如果你想在循环体内访问每个元素的索引，可使用内置的 enumerate 函数：

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: cat", "#2: dog", "#3: monkey", each on its own line
```

列表推导：在编程的时候，我们经常会想要将一种数据类型转换成另一种。下面是一个简单例子，将列表中的每个元素变成它的平方：

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)   # Prints [0, 1, 4, 9, 16]
```

使用列表推导，你可以让代码简化很多：

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Prints [0, 1, 4, 9, 16]
```

列表推导还可以包含条件：

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

表 1: 列表包含的函数

cmp(list1, list2)	比较两个列表的元素
len(list)	列表元素个数
max(list)	返回列表元素最大值
min(list)	返回列表元素最小值
list (seq)	将元组转换为列表

表 2: 列表包含的方法

list.append(obj)	在列表末尾添加新的对象
list.count(obj)	统计某个元素在列表中出现的次数
list.extend(seq)	在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
list.index(obj)	从列表中找出某个值第一个匹配项的索引位置
list.insert(index, obj)	将对象插入列表
list.pop(obj=list[-1])	移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
list.remove(obj)	移除列表中某个值的第一个匹配项
list.reverse()	反向列表中元素
list.sort([func])	对原列表进行排序

1.4.2 字典（Dictionaries）

字典用来存储（键，值）对。你可以这样使用它：

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data
print(d['cat'])       # Get an entry from a dictionary; prints "cute"
print('cat' in d)     # Check if a dictionary has a given key; prints "True"
d['fish'] = 'wet'     # Set an entry in a dictionary
print(d['fish'])      # Prints "wet"
# print(d['monkey'])  # KeyError: 'monkey' not a key of d
print(d.get('monkey', 'N/A'))  # Get an element with a default; prints "N/A"
print(d.get('fish', 'N/A'))    # Get an element with a default; prints "wet"
del d['fish']         # Remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

想要知道字典的其他特性，请查阅文档。

循环（Loops）：在字典中，用键来迭代更加容易。

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

如果你想要访问键和对应的值，那就使用 items 方法：

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

字典推导（Dictionary comprehensions）：和列表推导类似，但允许你方便地构建字典。

```
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)  # Prints "{0: 0, 2: 4, 4: 16}"
```

1.4.3 集合（Sets）

sets 是离散元的无序集合。示例如下：

表 3: 字典包含的函数

cmp(dict1, dict2)	比较两个字典的元素
len(dict)	计算字典元素个数，即键的总数
str(dict)	输出字典可打印的字符串表示
type(variable)	返回输入的变量类型，如果变量是字典就返回字典类型

表 4: 字典包含的方法

dict.clear()	删除字典内所有元素
dict.copy()	返回一个字典的浅复制
dict.fromkeys(seq[, val]))	创建一个新字典，以序列 seq 中元素做字典的键，val 为字典所有键对应的初始值
dict.get(key, default=None)	返回指定键的值，如果值不在字典中返回 default 值
dict.has_key(key)	如果键在字典 dict 里返回 true，否则返回 false
dict.items()	以列表返回可遍历的 (键, 值) 元组数组
dict.keys()	以列表返回一个字典所有的键
dict.setdefault(key, default=None)	和 get() 类似, 但如果键不存在于字典中，将会添加键并将值设为 default
dict.update(dict2)	把字典 dict2 的键/值对更新到 dict 里
dict.values()	以列表返回字典中的所有值
dict.pop(key[,default])	删除字典给定键 key 所对应的值，返回值为被删除的值。key 值必须给出。否则，返回 default 值
dict.popitem()	随机返回并删除字典中的一对键和值

```
animals = {'cat', 'dog'}
print('cat' in animals)   # Check if an element is in a set; prints "True"
print('fish' in animals)  # prints "False"
animals.add('fish')       # Add an element to a set
print('fish' in animals)  # Prints "True"
print(len(animals))       # Number of elements in a set; prints "3"
animals.add('cat')        # Adding an element that is already in the set does
nothing
print(len(animals))       # Prints "3"
animals.remove('cat')     # Remove an element from a set
print(len(animals))       # Prints "2"
```

和前面一样，要知道更详细的，查看文档。

循环（Loops）：在集合中循环的语法和在列表中一样，但是集合是无序的，所以你在访问集合的元素的时候，不能做关于顺序的假设。

```
animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: fish", "#2: dog", "#3: cat"
```

集合推导（Set comprehensions）：和列表推导及字典推导类似，可以很方便地构建集合：

```
from math import sqrt
```

```
nums = {int(sqrt(x)) for x in range(30)}
print(nums)  # Prints "{0, 1, 2, 3, 4, 5}"
```

1.4.4 元组（Tuples）

元组是一个值的有序列表（不可改变）。从很多方面来说，元组和列表都很相似。和列表最重要的不同在于，元组可以在字典中用作键，还可以作为集合的元素，而列表不行。例子如下：

```
d = {(x, x + 1): x for x in range(10)}  # Create a dictionary with tuple keys
t = (5, 6)        # Create a tuple
print(type(t))    # Prints "<class 'tuple'>"
print(d[t])       # Prints "5"
print(d[(1, 2)])  # Prints "1"
```

表 5: 元组的内置函数

cmp(tuple1, tuple2)	比较两个元组的元素
len(tuple)	计算元组元素个数
max(tuple)	返回元组中元素最大值
min(tuple)	返回元组中元素最小值
tuple(seq)	将列表转换为元组

文档有更多关于元组的信息。

1.5 函数

Python 中使用 def 关键字来定义函数，如：

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(sign(x))
# Prints "negative", "zero", "positive"
```

我们常常使用可选参数来定义函数，例如：

```
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True)  # Prints "HELLO, FRED!"
```

1.6 类

Python 中，对类的定义是简单直接的：

```
class Greeter(object):

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance method
    def greet(self, loud=False):
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)

g = Greeter('Fred')  # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

更多类的信息请查阅文档。