Python

覃雄派

Python

提纲

- Python语言简介
- Python编程环境安装
- Python语言入门
  - 常量、变量、注释
  - 数据类型
  - 运算符、及其优先级、表达式
  - 顺序、分支、循环
  - 函数
  - 面向对象编程
  - 异常处理
  - 正则表达式
- 常用Python库

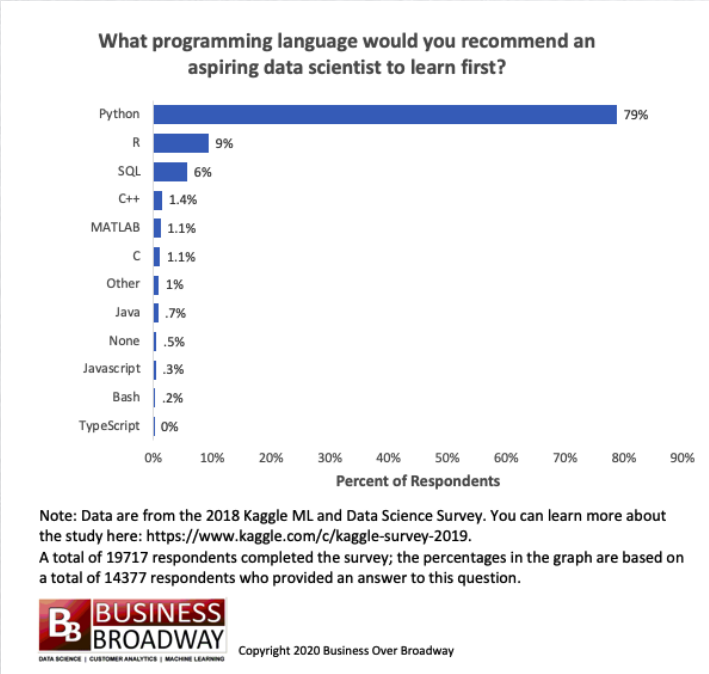
Python

- Data Scientist
- “Data Scientist = statistician who uses **python** and lives in San Francisco”

Python

- Python：数据科学家的最爱

What language would you recommend new data scientists learn first?

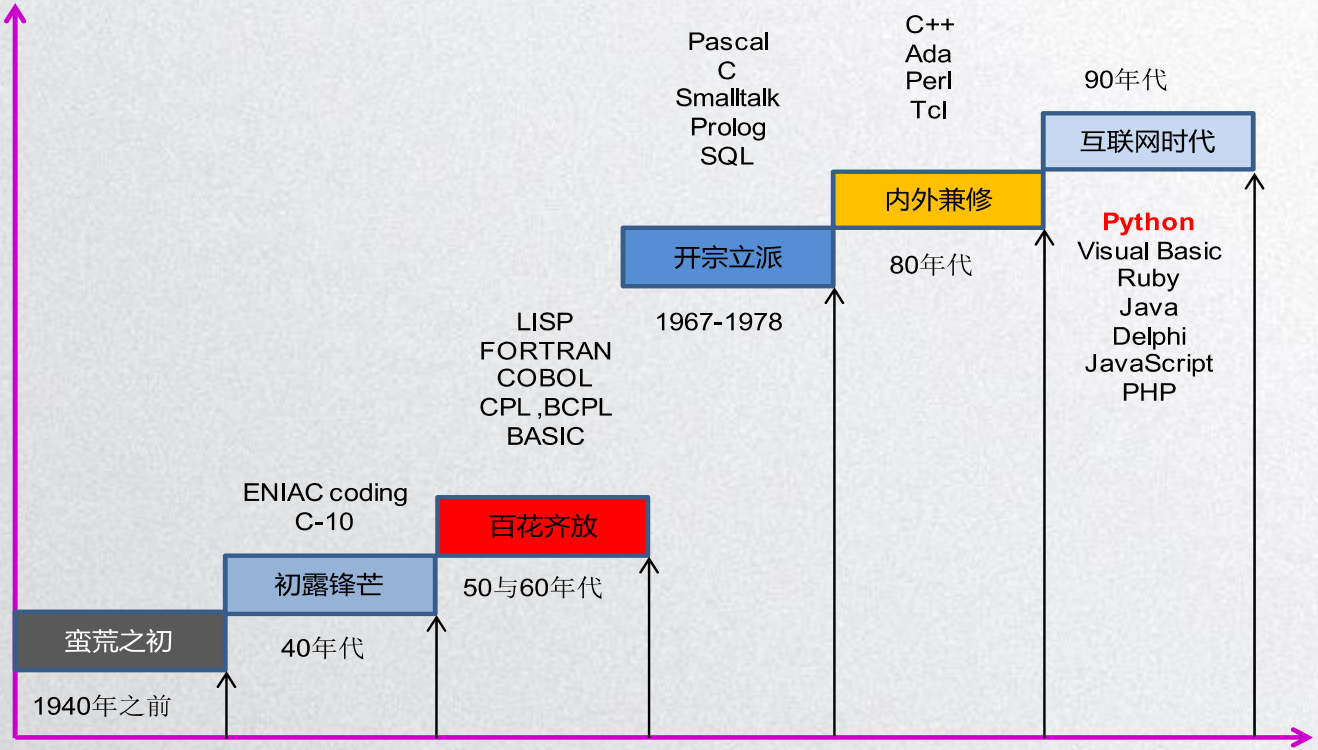


Every data scientist has an opinions on what language you should learn first. As it turns out, people who solely use Python or R feel like they made the right choice. But if you ask people that use both R and Python, they are twice as likely to recommend Python.

http://businessoverbroadway.com/wp-content/uploads/2020/06/Kaggle_Prog_First_2019.png

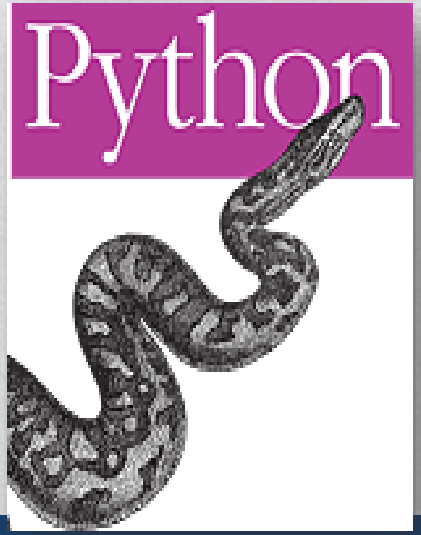
Python

- 计算机程序设计语言的历史



Python

- Python简介
  - Python 是20世纪八十年代末（1989年），在荷兰国家数学和计算机科学研究所设计出来的一种程序设计语言
  - 创始人为吉多·范罗苏姆（Guido van Rossum）



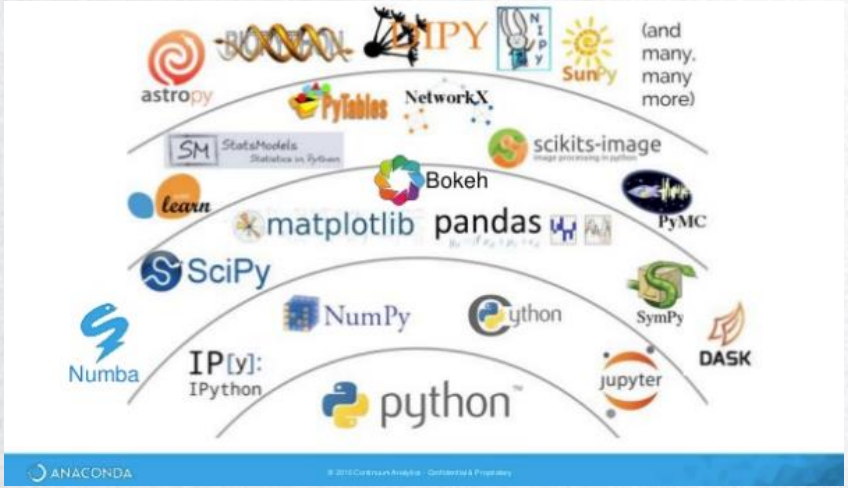
Python

- Python的优势
  - 胶水语言（Python as Glue）
  - 强大的数据科学生态
  - Solving the “Two-Language” Problem

This "two-language" problem is a trade-off that developers typically make when choosing a language -- it can either be relatively **easy for humans to write**, or **relatively easy for computers to run**, but not both.

- 何时不用Python
  - 性能要求很强的场景
    - 低延迟、高并发
    - Python支撑“双十一”？

得益于人们开发的各种数据预处理、数据挖掘与机器学习、自然语言处理、数据可视化等软件库，Python的应用领域得到了扩展，被应用到各种数据分析场合



https://pydsc.files.wordpress.com/2017/11/pythonenvironment.png?w=663

Python

- Python简介
  - 解释型语言与编译型语言?

请问Python属于：

A. 编译型语言

B. 解释型语言

Python虚拟机本身，几乎可以在所有的操作系统中运行；所以，我们可以在几乎所有的操作系统上，运行Python程序，包括Windows、Linux、Unix、Mac OS等

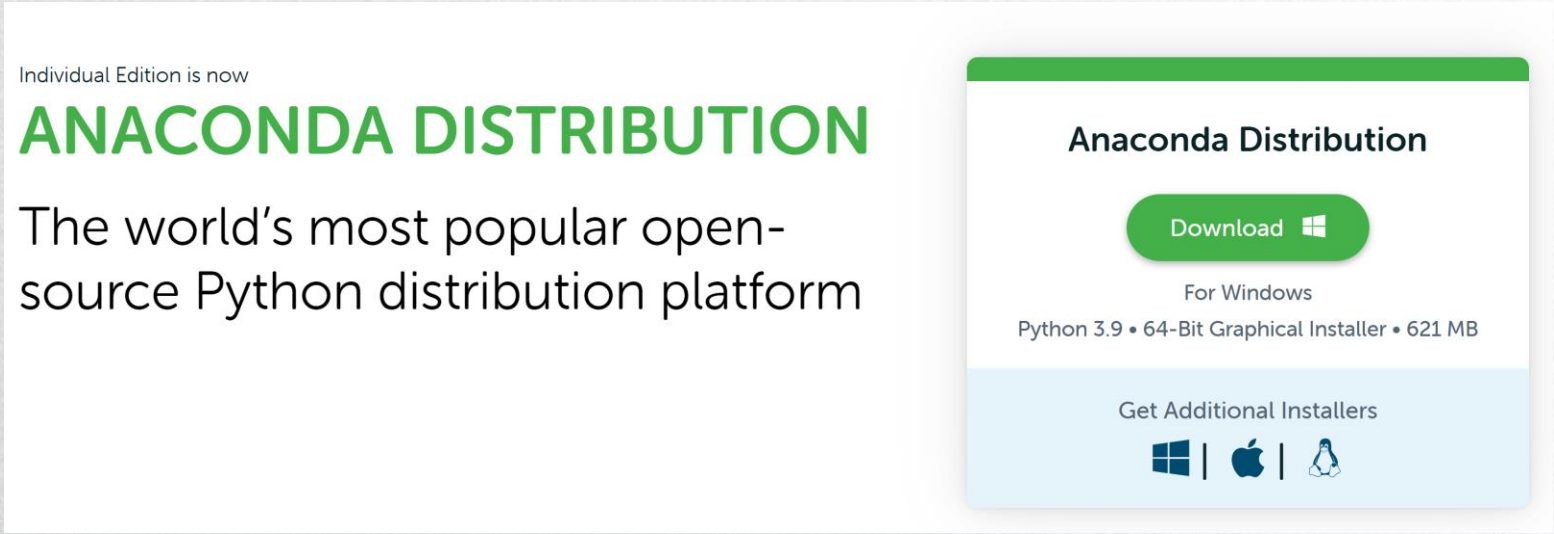
Python

Python

- Python 编程环境
  - 虽然Python易学易用，但两个难题经常让人是否头痛：
    - 不同版本问题：Python 3.X
    - Python包管理：从哪里下载numpy和pandas?
  - 使用Anaconda进行Python开发
    - 在同一个机器上安装不同版本的软件包及其依赖
    - 能够在不同的环境之间进行切换

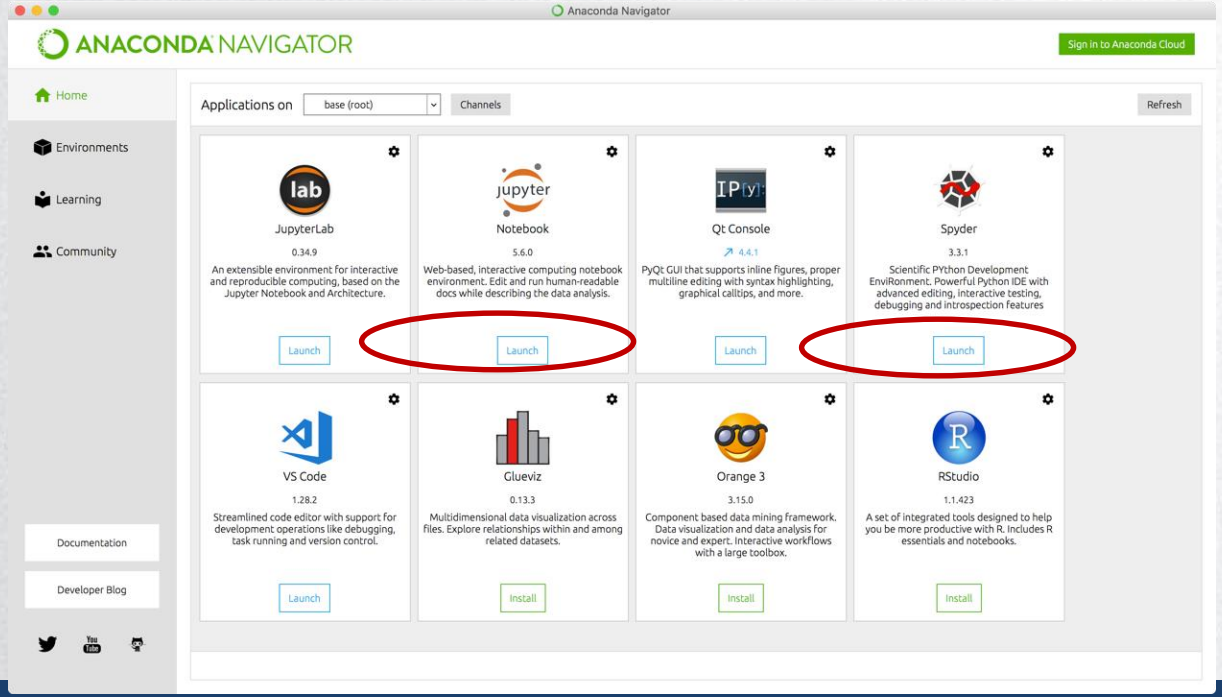
Python

- Anaconda环境安装
  - 登录Anaconda官网：https://www.anaconda.com/distribution/



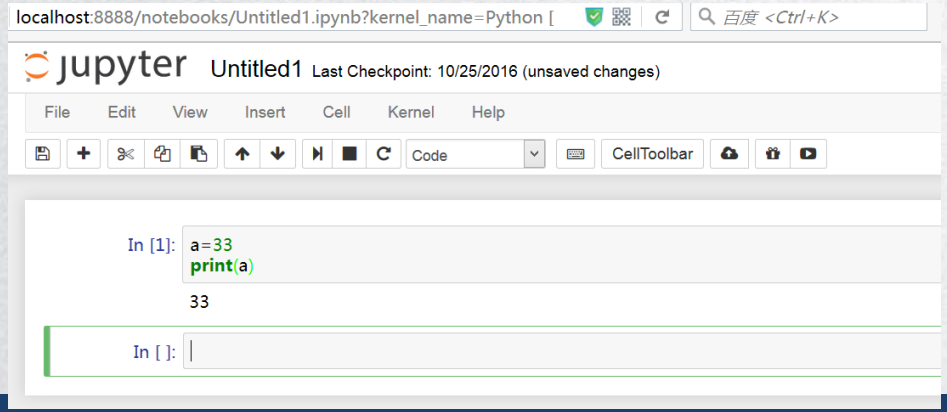
Python

- Python安装
  - 通过Navigator启动Jupyter或者Spyder



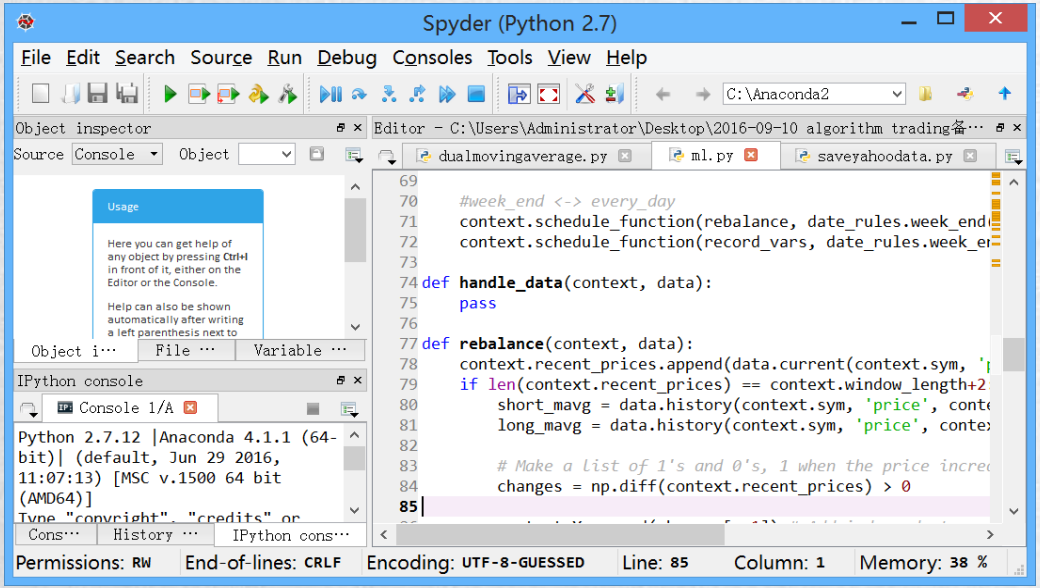
Python

- 交互式编程Jupyter
  - 启动Jupyter Notebook的过程(Windows)是，点击Windows开始菜单，找到Anaconda3程序组，点击“运行Jupyter Notebook”快捷方式，等待一会，浏览器自动启动，展示Jupyter Notebook界面
  - 然后选择“New”菜单，选择“Python[Root]”菜单项，新建Notebook
  - 然后用户就可以在输入框，按照Python的语法要求，输入Python代码，点击运行按钮，解释执行代码，即可立即观察到执行的结果



Python

- 开发复杂项目
  - 启动Spyder开发环境的办法是，点击Windows开始菜单，找到Anaconda3程序组，点击“运行Spyder”快捷方式
  - Spyder启动以后，用户可以新建项目、新建文件、编写代码、和调试代码



Python

Python

- Python语言基础：变量、常量、注释
  - 变量是在程序的执行过程中可以改变的量，它可以是任意的数据类型
  - 在Python中，变量无需事先定义其数据类型，给其进行赋值的时候，就确定了它的数据类型
    - 变量需要有一个名称，称为变量名
      - 变量名以英文、_开头，后续字符可以是英文、数字、或者_
      - 比如my_book是一个有效的变量名，而9book则不是一个有效的变量名

```
In [2]: a=35
        b=76.7
        c="I am a boy"
        print(a,b,c)

        (35, 76.7, 'I am a boy')
```

Python

- Python语言基础：变量、常量、注释
  - 常量是在程序的执行过程中不能改变的量，包括整数、小数、字符串等常量
    - 比如35是一个整数常量，而76.7则是一个小数常量，而“I am a boy”是一个字符串常量

```
In [2]: a=35
        b=76.7
        c="I am a boy"
        print(a,b,c)

        (35, 76.7, 'I am a boy')
```

Python

- Python语言基础：变量、常量、注释
  - Python的注释以#号开头，#号之后一直到一行末尾的所有字符，都是注释的一部分，Python解释器将忽略它们，不予执行
    - 下面的代码里，this is a comment和this is another comment都是注释

```
In [3]: #this is a comment
        name = "mark" #this is another comment
        print(name)

        mark
```

Python

- Python语言基础：数据类型
  - **布尔值bool**
  - 布尔值只有True、False两种值，要么是True，要么是False
  - 在Python中，可以直接用True、False表示布尔值，也可以通过关系运算或者逻辑运算计算出来
    - 下面的代码中，通过关系运算3>2(为真，即True)，给变量b1赋值。

```
In [4]: b1 = 3>2
        print(b1)
        type(b1)   #输出b1的数据类型，也就是bool

        True

Out[4]: bool
```

Python

- Python语言基础：数据类型
  - **整数int**
  - Python可以表示精度不限的整数(包括0和负整数)
  - 在程序中，整数的表示方法和数学上的写法是一致的，比如100、-100、0等
    - 下面的代码中，给整数变量a、b、c进行了赋值

```
In [5]: a=100
        b=-100
        c=0xff
        type(c)   #输出c的数据类型，也就是int

Out[5]: int
```

0xff为十六进制整数

Python

- Python语言基础：数据类型
  - **浮点数float**
  - 浮点数也就是小数
  - 浮点数可以使用标准的写法，如3.23、3.24、-9.11等，也可以使用科学计数法表示，比如3.23e3表示3.23*103，即3230，1.2e-5表示0.000012
    - 下面的代码中，分别给变量f1、f2进行了浮点数赋值

```
In [6]: f1=3.23
        f2=1.2e-5
        type(f1)   #输出f1的数据类型，也就是complex

Out[6]: float
```

Python

- Python语言基础：数据类型
  - **复数complex**
  - 除了整数和小数，Python还支持复数，复数包括实部和虚部，虚部带一个字符j
    - 下面的代码，分别给变量c1和c2进行了复数赋值。

```
In [7]: c1=3+2.7j
        c2=-5-9.3j
        type(c1)  #输出c1的数据类型，也就是complex

Out[7]: complex
```

Python

- Python语言基础：数据类型
  - **字符串str**
  - 字符串是由若干字符组成的有序序列，字符串可以用单引号或者双引号括起来
  - 如果分别用三个双引号首尾括起来，可以用若干行的字符串常量，给一个字符串变量赋值，换行和空格都是字符串的一部分
  - 下面代码中，分别给变量s1、s2、s3和s4进行了字符串赋值。并且通过下标寻访其元素(即字符)
  - 下标的各种使用方式，代码中给出了说明
    - 注意Python的下标是从0开始的

```
In [8]: s1='small string'
        s2="a much larger sting"
        s3="""spanning
        multiple
        lines"""
        print(s3)
        type(s1)  #输出s1的数据类型，也就是str

        str1 = 'Hello World!'
        print str1      # 打印整个字符串
        print str1[0]    # 打印0号下标字符
        print str1[2:5]   # 打印2、3、4号下标字符
        print str1[2:]    # 打印2号下标字符，以及后续字符
        print str1 * 2    # 打印字符串两遍
        print str1 + "TEST" # 把str和"TEST"连接起来，然后打印

spanning
multiple
lines
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

Python

- Python语言基础：数据类型
- **列表list**
  - 列表是包含多种不同类型的元素的、可以改变的有序序列，当然列表的元素也可以是同样类型的，比如整数列表、小数列表等
  - 列表的表示方法是，用[]把元素包含起来，中间用逗号隔开
  - 下面的代码中，给列表list进行了赋值，并且通过下标寻访其元素
  - 下标的各种使用方式，代码中给出了说明

```
In [9]: list_normal = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
        tinylist = [123, 'john']

        print list_normal     # 打印list的所有元素
        print list_normal[0]    # 打印list的0号下标元素
        print list_normal[1:3]   # 打印下标为1、2的元素
        print list_normal[2:]   # 打印下标>=2的元素
        print list_normal[-1]   # 打印倒数第一个元素
        print list_normal[:-1]   # 从下标为0的元素开始打印，打印到倒数第2个元素

        print tinylist * 2  # 打印tinylist两遍
        print list_normal + tinylist # 连接list和tinylist，然后打印

['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
70.2
['abcd', 786, 2.23, 'john']
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Python

- 下标范围总结
  - 列表、元组、数组等

元素	33	55	44	88	77
顺序下标	0	1	2	3	4
逆序下标	-5	-4	-3	-2	-1

下标范围	含义
[0]	0号下标元素
[1:3]	1、2等下标对应子序列，注意不包括下标3
[1:]	1、2、3、4等下标对应子序列
[:3]	0、1、2等下标对应子序列，注意不包括下标3
[-1]	倒数最后一个元素
[:-1]	从头开始，到倒数最后一个元素之前的元素，注意不包括下标-1，构成的子序列
[0:5:2]	0、2、4等下标对应的子序列，注意步长为2，注意不包括下标5

Python

- Python语言基础：数据类型
- **列表list**
  - 列表是包含多种不同类型的元素的、可以改变的有序序列，当然列表的元素也可以是同样类型的，比如整数列表、小数列表等

请给出右侧题目的正确选项

```
In [3]: list1 = ['abcd',789,2.23,'john',70.2]
        print(list1)

        ['abcd', 789, 2.23, 'john', 70.2]

In [4]: print(list1[0])

        abcd

In [5]: print(list1[1:3])
```

A. [789, 2.23, 'john']　　B. [789, 2.23]

```
In [6]: print(list1[-1])

        70.2

In [7]: print(list1[:-1])
```

A. ['abcd', 789, 2.23, 'john']　　B. [70.2]

Python

- Python语言基础：数据类型
- **元组tuple**：元组也是一种有序序列，和列表非常相似
  - 元组和列表的主要区别：元组一旦初始化后，就不能修改了
  - 它也没有append()、insert()这样的方法。元组可以被看作是只读的**(read-only)**列表
  - 由于元组的元素是不可变的，所以用户编写的代码更加安全，不会由于不小心修改了某些元素，而导致程序执行错误

```
tuple1 = ('abcd',789,2.23,'john',70.2)
print(tuple1[0])
print(tuple1[1:3])
print(tuple1[-1])
print(tuple1[:-1])
```

```
abcd
(789, 2.23)
70.2
('abcd', 789, 2.23, 'john')
```

Python

- Python语言基础：数据类型
- **字典dict**
  - Python的字典，实际上是一个哈希表(Hash Table)，这个表包含一系列的键值对(Key Value Pairs)，键值对的key和value可以是整数、小数、字符串、或者布尔值等数据类型；dict具有极快的查找速度

```
dict1 = {'name':'john', 'code':6734, 'dept':'cs',\
         'score':90.0, 1:'aux'}
print(dict1)
print(dict1[1])
print(dict1['code'])
print('code' in dict1.keys())
```

```
{'name': 'john', 'code': 6734, 'dept': 'cs', 'score': 90.0, 1: 'aux'}
```

A	B
6734 6734 0	aux 6734 True

请给出左侧题目后3个操作的正确选项

Python

- Python语言基础：运算符及其优先级、表达式
  - Python的运算符包括算术运算符、关系运算符、逻辑运算符、集合运算符、对象运算符等。
  - 运算符是有优先级的，比如在一个表达式中，既有加减法运算，又有乘除法运算，那么在没有括号的情况下，先做乘除法运算，再做加减法运算，而使用括号，则改变运算执行的顺序
    - 比如a+b*c，对该表达式进行计算的时候，先做b*c，再把结果和a相加，得到最终结果
    - 如果把该表达式改成(a+b)*c，那么对该表达式进行计算的时候，先做a+b，再把结果和c相乘，得到最终结果

Python

- Python语言基础：运算符及其优先级、表达式
  - 在变量、常量、运算符的基础上，我们就可以构造表达式，对数据进行计算
  - 表达式是利用运算符，把兼容的常量、变量拼接起来的式子，表达式是编写程序的基础。
    - 比如我们有两个整数类型的变量，那么就可以利用关系运算符，构造关系运算表达式a>b，当a的值大于b的时候，其值为真(True)，否则为假(False)
    - 还可以在此基础上，构造逻辑表达式，比如a>b and c>d，那么当a>b和c>d同时为真的时候，该表达式的值为真

Python

- Python语言基础：顺序、分支、循环
  - **顺序程序结构**
  - 顺序结构是最简单的一种结构
  - 解释程序执行顺序结构的Python程序时，它将顺序地解释执行各个语句，直到程序的末尾
  - 下面的程序，首先给两个变量赋值，然后交换两个变量，最后打印出两个变量的值

```
In [15]: #顺序结构
         a=3
         b=4

         t=a
         a=b
         b=t
         print(a)
         print(b)

         4
         3
```

Python

- Python语言基础：顺序、分支、循环
  - **分支程序结构**
  - 分支程序结构用于根据一定的条件进行判断(关系表达式、逻辑表达式)，然后决定程序的走向
  - 它由if语句、else语句、elif语句来构造，分支程序可以嵌套
  - 我们通过实例来了解if语句的语法结构和功能
  - 分支程序结构的第一个实例如下，当a的值大于b的时候，对其值进行交换，最后先输出a，再输出b，也就是按照从小到大的顺序输出b

注意缩进与对齐...

```
In [16]: #分支结构
         a=3
         b=2
         if a>b:
             t=a
             a=b
             b=t#这三个语句属于一个语句块，注意语句块的缩进
         print(a)
         print(b)

         2
         3
```

Python

- Python语言基础：顺序、分支、循环
  - **分支程序结构**
  - 分支程序结构的第二个实例如下，当a的值大于b，则打印a greater than b，否则打印a less than or equal to b
  - 在这个实例里面，条件为真的时候，我们要进行某种处理，条件为假的时候，要进行另外一种情况的处理

```
In [17]: a=3
         b=2
         if a>b:
             print('a greater than b')
         else:
             print('a less than or equal to b')

         a greater than b
```

Python

- Python语言基础：顺序、分支、循环
  - **分支程序结构**
  - 分支程序结构的第三个实例如下，这个实例对不同区段的成绩，进行A、B、C、D、E级别的分档
  - 需要使用if…elif…else分支程序结构

```
In [18]: score=75
         grade='A'
         if (score<60):
             grade='E'
         elif (score<70):
             grade='D'
         elif (score<80):
             grade='C'
         elif (score<90):
             grade='B'
         else:
             grade='A'
         print(grade)

         C
```

Python

- Python语言基础：顺序、分支、循环
  - **循环程序结构**
  - 我们可以用两个关键字while和for来构造循环程序结构
    - 循环程序结构的第一个例子是一个while循环，首先给变量i赋予初值1，然后通过while循环判断它是否还在1到5之间，然后把i累加到变量sum中，最后求出1+2+3+4+5的值

```
In [19]: #循环结构
         i=1
         sum=0
         while( i<=5):
             sum = sum +i
             i = i +1
         print(sum)

15
```

Python

- Python语言基础：顺序、分支、循环
  - **循环程序结构**
  - 循环程序结构的第二个实例是一个for循环
  - 首先创建一个列表，然后对于列表长度(为3)之上创建的一个有效下标范围(0、1、2)的每个下标，顺序访问列表的每个元素

```
In [20]: fruits = ['banana', 'apple', 'mango']
         for index in range(len(fruits)):
             print 'Current fruit :', fruits[index]

         print "Good bye!"

Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

- len(fruits)返回列表长度，即3
- range(3)返回[0,1,2]列表

Python

- Python语言基础：函数、库函数
  - 函数是从英文的“function”直接翻译过来的
  - 在Python语言里，函数是具有一定功能的一段代码
    - 对于经常用到的一些功能，比如打印输出变量的值，可以把实现这些功能的代码组织成函数的形式
    - 在需要这些功能的时候，直接调用函数即可，而无需再写一遍类似的代码，函数有利于程序的模块化设计风格的实现
  - 定义函数的时候，我们就规定好函数接受什么样的参数，将返回什么样的值
    - 对于调用者来讲，只需要了解这些信息就足够了，至于函数内部是如何实现对应的功能的，他无需关心

Python

- Python语言基础：函数、库函数
- **内置函数的使用**
- Python解释器已经内置了若干函数，方便用户编程时调用。这些函数可以实现数学运算、集合操作、逻辑判断、输入输出等功能。
- 在这里，我们介绍print函数
- Python其它内置函数及其使用方法，请参考https://docs.python.org/2/library/functions.html。

- print语句主要用于输出用户数据，一般输出到屏幕上，即python对象sys.stdout
- print语句可以实现灵活的输出

```
In [2]: a =3
        b=4.5
        c='young man'
        print(a, b, c) #顺序输出a,b,c的内容
        print '%d,%f,%s\n'%(a,b,c)#按照格式串'%d,%f,%s\n'输出a,b,c的值
        #%d表示输出整数，%f表示输出小数，%s表示输出字符串，\n表示换行

        (3, 4.5, 'young man')
        3,4.500000,young man
```

Python

- Python语言基础：函数
  - 用户自定义函数
  - 下面的代码，展示了二分查找函数的定义，以及对它的两次调用
  - 从这个实例可以看出，通过把一些公用的功能实现为一个函数
  - 我们可以多次调用实现更加复杂的功能，代码则变得简洁多了

In [23]:

```
def bin_search(a, target):
    low = 0
    high = len(a) - 1

    while low <= high:
        mid = (low + high) // 2
        midVal = a[mid]

        if midVal < target:
            low = mid + 1
        elif midVal > target:
            high = mid - 1
        else:
            return mid
    return -1

a = [1,2,3,4,5,6,7,8]
target = 6
found_index = -1
found_index = bin_search(a,target)
if(found_index==-1):
    print('not found')
else:
    print '%s:%d'%('found', found_index)

a = [3,5,7,11,13,17,23,29]
target = 2
found_index = -1
found_index = bin_search(a,target)
if(found_index==-1):
    print('not found')
else:
    print '%s:%d'%('found', found_index)
```

```
found:5
not found
```

Python

- Python语言基础：函数与递归
  - 在Python中，在实现一个函数的时候，可以调用其它函数，甚至可以调用自身，即函数的递归调用
  - 函数的递归调用，使得解决一些问题的代码变得简洁、易于理解。
    - 采用函数的递归调用，计算n的阶乘，其设计思路是，(1) 如果n==0或者n==1，那么n的阶乘为1。(2) 如果我们知道了n-1的阶乘，把它乘上n就可以得到n的阶乘，问题的规模就缩小了一阶，也就是n的阶乘的计算变成n-1阶乘的计算，加上一个附加的步骤(乘上n)
    - 由于0或者1的阶乘，我们是很容易得到的，也就是问题规模缩小到1或者0的时候，我们就可以解了，一旦低阶的问题得到解决，那么我们就可以一步步倒退回去，把各个更高阶的问题解决掉

```
In [24]: def fractal(n):
             if n==1 or n==0:
                 return 1
             return n*fractal(n-1)

         print fractal(5)
```

```
120
```

Python

- Python语言基础：类和对象
  - Python是一种面向对象的编程语言，它通过封装机制，把数据和对数据的操作，封装成类
  - 而类的实例化则是一个个的对象
  - 比如，我们要对职员进行管理，需要登记他们的姓名、性别、年龄、薪水等信息，针对某个职员，可以显示他的这些信息
  - 我们通过设计职员类，把上述属性管理起来，并且提供显示职员信息的操作函数
  - 职员类建立以后，我们可以生成职员类的实例，分别对应John、Mary等职员。

In [3]:
```
class Employee(object):
    def __init__(self, _name):
        self.name = _name

    def setName(self, _name):
        self.name = _name
    def setSex(self, _sex):
        self.sex = _sex
    def setAge(self, _age):
        self.age = _age
    def setSalary(self, _salary):
        self.salary = _salary
    def show(self):
        print 'name:', self.name, ', sex:', self.sex, ', age: ',self.age, ', salary: ',self.salary

emp1 = Employee('John')
emp1.setName('John')
emp1.setSex('Male')
emp1.setAge(33)
emp1.setSalary(9800.00)
emp1.show()

emp2 = Employee('Mary')
emp2.setName('Mary')
emp2.setSex('Female')
emp2.setAge(35)
emp2.setSalary(9500.00)
emp2.show()
print Employee.__name__ #显示类名
```

```
name: John , sex: Male , age:  33 , salary:  9800.0
name: Mary , sex: Female , age:  35 , salary:  9500.0
Employee
```

Python

- Python语言基础：类和对象
- **构造函数**
  - 一个类的构造函数负责对象的构造，构造函数的名称为__init__ ，它带一个self参数以及其它参数，其中self参数指向将要构造的对象，也就是self是对象的引用
  - 构造函数的作用是对对象进行初始化
  - 比如，上述代码中，Employee类的构造函数__init__，通过_name参数，给对象的name属性进行了赋值
  - 而对__init__函数的调用隐含在emp1 = Employee('John')、和emp2 = Employee('Mary')语句的调用过程中，这两个语句分别创建了emp1对象和emp2对象，它们的name属性分别为'John'和'Mary'
- **对象的摧毁和垃圾回收**
  - 对于程序不再使用的对象，Python周期性执行垃圾回收过程，自动删除这些对象，以释放它们占用的内存空间

Python

- Python语言基础：类和对象
  - **继承**
  - 在定义类的时候，我们可以基于已有的类定义新的类，新的类(子类)和已有的类(父类)是继承的关系
  - 子类继承了父类的所有属性和方法(函数)，还可以增加新的属性和方法
  - 比如，我们定义了一个新的类Manager，它继承于Employee类，但是增加了一个新的属性subsidy(特殊津贴)

```
In [4]: class Manager(Employee): # define child class
            def setSubsidy(self, _subsidy):
                self.subsidy = _subsidy

        mgr1 = Manager('tom')
        mgr1.setName('tom')
        mgr1.setSex('Male')
        mgr1.setAge(31)
        mgr1.setSalary(10100.00)
        mgr1.setSubsidy(1000.00)
        mgr1.show()

        name: tom , sex: Male , age:  31 , salary:  10100.0
```

Python

- Python语言基础：类和对象
- **重写Override**
  - 在上一个实例中，mgr1.show()只显示了经理的姓名、性别、年龄和薪水，但是没有显示特殊津贴
  - 为此，我们为新的类Manager定义一个新的show函数，这个函数和父类Employee的show函数同名，但是功能有些不一样，除了显示姓名、性别、年龄和薪水，它还显示特殊津贴
    - 这种对父类的方法进行重新定义的机制，称为重写Override

In [5]:
```
class Manager(Employee): # define child class
    def setSubsidy(self, _subsidy):
        self.subsidy = _subsidy
    def show(self):
        print 'name:', self.name, ', sex:', self.sex, ', age: ',self.age, ', salary: ',self.salary, ', subsidy: ',self. subsidy

mgr1 = Manager('tom')
mgr1.setName('tom')
mgr1.setSex('Male')
mgr1.setAge(31)
mgr1.setSalary(10100.00)
mgr1.setSubsidy(1000.00)
mgr1.show()
```

```
name: tom , sex: Male , age:  31 , salary:  10100.0 , subsidy:  1000.0
```

Python

- Python语言基础：异常处理
  - 程序执行过程中，可能发生异常情况，比如一个非零的整数除以0就会发生异常
    - 我们可以捕抓异常，然后打印提示信息，帮助用户了解到发生的情况
    - 用户可以采取补救措施，比如等待用户输入正确的数值、释放磁盘空间、连接到互联网等
    - 一般把有可能引发异常的代码放在一个try:语句块里，在try:语句块之后，跟着一个except:语句及其语句块，该语句块的代码，将对错误情况作出处理

In [6]:
```
try:
    a = 10
    b = 0
    print a / b
except ZeroDivisionError:
    print 'divide by zero error captures! '
```

```
divide by zero error captures!
```

In [7]:
```
class Networkerror ( RuntimeError):
    def __init__(self, _args):
        self.args = _args

try:
    host_not_found = True
    if (host_not_found):
        raise Networkerror("host not found")
        #如果没有异常，继续执行后续代码……
except Networkerror, e:
    print e.args
```

```
('h', 'o', 's', 't', ' ', 'n', 'o', 't', ' ', 'f', 'o', 'u', 'n', 'd')
```

Python

- Python语言基础：正则表达式
- 正则表达式，是一种用来匹配字符串的有力工具
- 它是一个特殊的字符序列，用于匹配或者查找其它字符串里面的子串
  - 正则表达式可以用来判断一个字符串是否日期、电子邮件、邮政编码、或者电话号码，帮助我们对用户输入数据，进行合法性检验，也可以用来在一个字符串里面寻找这些实体类型
  - 下面的实例，用正则表达式匹配电话号码。正确的电话号码的模式是，3个数字跟着一个横杠、然后跟着3个数字，再跟着一个横杠，最后跟着4个数字，具体的模式是'^(\d{3})-(\d{3})-(\d{4})$'，^表示开始，$表示结束，\d表示数字
  - 这个正则表达式经过编译以后，就可以用来匹配电话号码

```
In [8]: import re

        #正则表达式匹配电话号码
        pattern_phone=re.compile('^(\d{3})-(\d{3})-(\d{4})$')

        phone='800-555-1212'
        phonematch=pattern_phone.match(phone)
        if phonematch:
            print phonematch.group()
        else:
            print "phone number is error!"

        phone='800-555-1212-1234'
        phonematch=pattern_phone.match(phone)
        if phonematch:
            print phonematch.group()
        else:
            print "phone number is error!"

800-555-1212
phone number is error!
```

Python

- Python语言基础：正则表达式

下面哪个字符串可以匹配正则表达式 (4{5,6})

A. 456

B. 44444 or 444444

C. 4444444444 or 444444444444

“文本采集”部分，专门介绍正则表达式

无需死记硬背，可以登录如下网站测试各种正则表达式
https://www.regextester.com/

Python

Python

- 常用Python库
  - Pandas：二维表处理
  - Numpy/SciPy：数组处理
  - Scikit-learn：传统机器学习
  - Keras & tensor flow：深度学习
  - Matplotlib：数据可视化
  - networkX：图数据分析
  - NLTK/ Gensim：自然语言处理