前面经过十几篇文章，相想必大家对于数据分析是什么，怎么做有了基本的认识。

跟着操作的小伙伴基本功应该练的差不多了，可以蛟龙出海了。

有了前面的基础，理论可以放一放，本周开始我们要学 Python，用 Python 做数据分析。

作为当下最热门的编程语言之一，Python 有两个非常有趣的方向：一个是数据分析，从掌握数据分析的基本方法开始，学习 NumPy、Pandas、mapplotlib 包；然后再往下就是数据挖掘，机器学习、深度学习，甚至人工智能。另外一个方向则是 web 开发。有同学说爬虫呢，爬虫其实是获取数据的一个手段，包括数据库的处理等等都是包含在上面两条路线里面。

想学会一门语言不是一朝一夕的事情，本文是按照业务数据分析师/商业分析师的路线来讲 Python 的学习路径。若大家想成为技术型的分析师，或者未来往数据挖掘发展，建议你要比文章内容学得更深，所有的代码最好都手打一遍，这是最有效的学习方式。

好了，言归正传。按照所有编程语言的学习套路，先从基础语法开始，有编程基础的童鞋可能学习起来比较轻松，但也建议看一遍，温习一二。

数据分析环境

Python 的编写环境，用 Anaconda 足矣。Anaconda 是专业的数据科学计算环境，已经集成绝大部分包和工具，不需要多余的安装和调试。

Python 版本建议 3.0 以上，不要选择 2.7 的版本，否则你会被无尽的中文编码问题困扰。

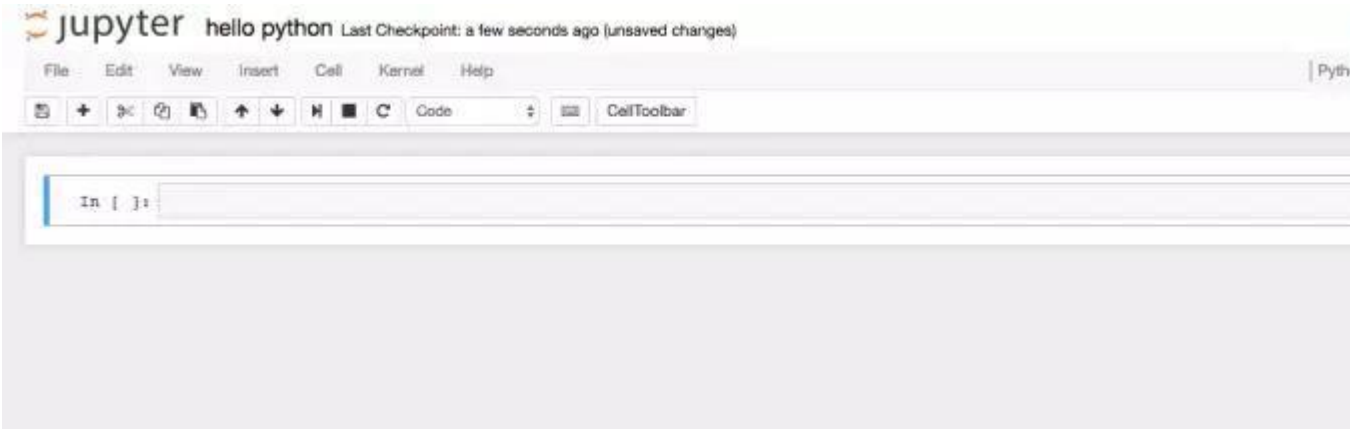
Anaconda 在官网下载，选择最新版本，约 400MB。

完成安装后，Win 版本会多出几个程序，Mac 版本只有一个 Navigator 导航。数据分析最常用的程序叫 Jupyter，以前被称为 IPython Notebook，是一个交互式的笔记本，能快速创建程序，支持实时代码、可视化和 Markdown 语言。

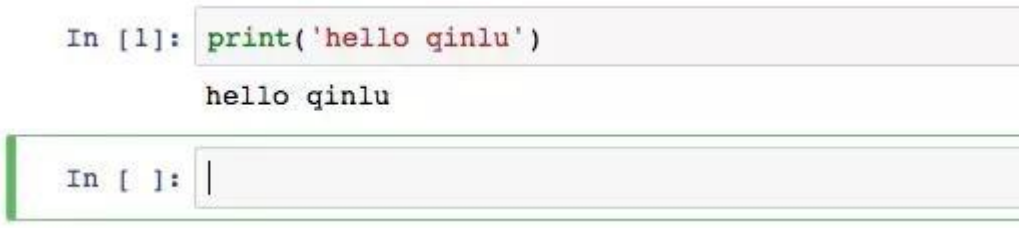
点击 Jupyter 进入，它会自动创建一个本地环境 localhost。

点击界面右上角的 new，创建一个 python 文件。

开始你的 Python



界面上部是工具栏，编辑撤回运行等，下面是快捷操作，大家以后会熟悉的。页面正中便是脚本执行的地方，我们输入自己第一行代码吧：



（我就不用 hello world）灰色框是输入程序的地方，回车是换行，shift+回车执行灰色区域的代码，它的结果会直接在下面空白处出现。这就是 Jupyter 交互式的强大地方，将 Python 脚本分成片段式运行，尤其适合数据分析的摸索调整工作。

这里的 print 叫函数，和 excel 的函数同理，是程序执行的主体，负责将输入转化成输出（函数留在下一篇细讲）。这里将 hello qinlu 这段文字输出。新手可能会奇怪为什么要加引号，这种用引号括起来的文字在程序中叫字符串。

Python 是一门计算机语言，它的逻辑和自然语言不一样，编程语言的目的是执行任务，所以它不能有歧义。为了规避各种歧义，人们创造了语法规则，只有正确的语法，才能被转换成 CPU 执行的机器码。

先了解 Python 语法中的数据类型。计算机最开始只被用于数值运算，后来被赋予了各种丰富的数据类型。

```
In [2]: 1+1
Out[2]: 2

In [3]: 1.1+1.2
Out[3]: 2.3
```

上面两个是小学生都会的四则运算，在计算机语言中可没有那么简单。它涉及了两个数值类型，整数 int 和浮点数 float。整数和浮点数在计算机内部存储的方式是不同的，我们不用知道具体原理，明确一点，整数运算是永远精确的，浮点运算则可能有误差。

两种数据类型也可以互换，通过 int 函数和 float 函数。

```
In [4]: float(1)+float(1)
Out[4]: 2.0

In [5]: int(1.1)+int(1.2)
Out[5]: 2
```

有了数值，必然有文本，程序中叫字符串，用英文引号括起来表示。单引号和双引号没有区别，所以"qinlu"和'qinlu'是等价的，引号是边界，输出的时候不会包含它。当字符串内本身包含引号时，也不影响使用。

```
In [8]: print("hello 'qinlu'")
        hello 'qinlu'

In [9]: print('hello "qinlu"')
        hello "qinlu"
```

需要注意的是，不论单引号还是双引号，一旦混用很容易出现错误。因为程序并不知道它是字符串的边界还是符号。

```
In [25]: print("a""b")
ab

In [26]: print(""b")
  File "<ipython-input-26-fa111d6e5f54>", line 1
    print(""b")
          ^
SyntaxError: EOF while scanning triple-quoted string literal
```

解决方法有两种，一种是使用三引号，三引号代表整体引用，而且包含换行。第二种是引号前面加\，它是转义字符，表示这个引号就是单纯的字符。

```
In [43]: print("\"\"b")
""b

In [44]: print("""
         'qin'
         ""
         """)

'qin'
""
```

三引号也可以用来注释，通常是大段的文字解释，如果一句话，我们更习惯用#，#后面的内容均不会作为程序执行。

时间是特殊的数值类型，它将结合 datetime 模块讲解。

还有两个常见的数据类型，布尔值和空值。布尔值是逻辑判断值，只有 True 和 False。

```
In [29]: 1<2
Out[29]: True

In [30]: 1>2
Out[30]: False
```

布尔值在 IF 语句和数据清洗中经常使用，利用其过滤。布尔值能和布尔值运算，不过这里是 and、not、or 作为运算符，Ttue and True = True，False and True = False，False and False = False， not True = False， True or False = True 等。

空值是一个特殊的值，表示为 None， None 不等于 0，0 具有数学意义而 None 没有， None 更多表示该值缺失。

整数，浮点数，字符串，布尔值，空值就是 Python 常见的数据类型。Python3 对中文的支持比较友好，所以大家可以用中文作为字符串试一下 print。

数据类型构成了变量的基础，变量可以是任意的数据类型。想要用变量，必须先赋予变量一个值，这个过程叫赋值。

```
In [45]: a = 1
         a
Out[45]: 1

In [47]: a = 'abc'
         a
Out[47]: 'abc'
```

我首先给 a 赋予了一个整数值 1，然后改变它为字符串 abc，变量在 Python 中没有固定的数值类型，这是 Python 最大的优点，所以它在数据分析中很灵活。这也是它被称为动态语言的原因，相对应的叫静态语言。

Python 是大小写敏感的语言，所以 a 和 A 是有区别的，这点请牢记。另外变量名尽可能使用英文，不要拼音，英文的可读性是优于拼音的。

变量有两种拼写风格，一种叫驼峰，一种叫下划线，以用户 ID 为例。驼峰命名法为 userId，以一串英文词语 user 和 id 组成变量，第一个词语的首字母小写，第二个词语开始的首字母均大写。下划线命名法为 user_id，全部小写，用_分割单词。

```
In [48]: a = 1
         b = a
         a
Out[48]: 1
```

一个变量的值可以被赋予另外一个变量，如果 b 变量之前有另外一个值，那么会被 1 覆盖。呈从上而下的执行关系。

```
In [49]: a = 1
         a = a + 1
         a

Out[49]: 2
```

初看 a = a + 1 好像有逻辑问题，其实这涉及到了程序执行的先后顺序，程序是先计算 a+1 的值得到 2，然后将其赋予(覆盖)了 a。等号右边的计算先于左边，这是从右到左的逻辑关系。

有变量，自然有常量，常量是固定不变的量，可是在 Python 中没有真正意义的常量，一切皆可变，它更多是习惯上的叫法，即一旦赋值，就不再改变了。

Python 的基础数学运算符号有+,-,*,/,//,%。前面四个就是加减乘除，其中除法的结果一定是浮点数。后面两个符号是除法的特殊形式，//代表除法中取整数，%代表除法中取余数。

```
In [53]: 7/2
Out[53]: 3.5

In [54]: 7//2
Out[54]: 3

In [55]: 7%2
Out[55]: 1
```

到这里，新手部分已经讲解完成。再来讲讲数据结构。

数据结构

Python 一共有三大数据结构，它是 Python 进行数据分析的基础，分别是 tuple 元组，list 数组以及 dict 字典。本文通过这三者的学习，打下数据分析的基础。

数组

数组是一个有序的集合，他用方括号表示。

```
In [2]: num = [1,2,3,4,5]
        num

Out[2]: [1, 2, 3, 4, 5]
```

num 就是一个典型的数组。数组不限定其中的数据类型，可以是整数也可以是字符串，或者是混合型。

数组可以直接用特定的函数，函数名和 Excel 相近。

```
In [3]: sum(num)
Out[3]: 15

In [4]: len(num)
Out[4]: 5
```

sum 是求和，len 则是统计数组中的元素个数。

上述列举的函数是数组内整体元素的应用，如果我只想针对单一的元素呢？比如查找，这里就要用到数组的特性，索引。索引和 SQL 中的索引差不多，都是用来指示数据所在位置的逻辑指针。数组的索引便是元素所在的序列位置。

```
In [5]: num[0]
Out[5]: 1

In [6]: num[1]
Out[6]: 2
```

注意，索引位置是从 0 开始算起，这是编程语言的默认特色了。num[0]指数组的第一个元素，num[1]指数组的第二个元素。

我们用 len()计算出了数组元素个数是 5，那么它最后一个元素的索引是 4。若是数组内的元素特别多呢？此时查找数组最后一位的元素会有点麻烦。Python 有一个简易的方法，可以用负数表示，意为从最后一个数字计算索引。

```
In [7]: num[4]
Out[7]: 5

In [8]: num[-1]
Out[8]: 5
```

这里的 num[4]等价于 num[-1]，num[-2]则指倒数第二个的元素。

再来一个新问题，如何一次性选择多个元素？例如筛选出数组前三个元素。在 Python 中，用：表示范围。

```
In [10]: num[0:3]
Out[10]: [1, 2, 3]
```

num[0:3]筛选了前三个元素，方括号左边是闭区间，右边是开区间，所以这里是 num[0]，num[1]和 num[2]，并不包含 num[3]。这个方法叫做切片。

```
In [11]: num[0:]
Out[11]: [1, 2, 3, 4, 5]

In [12]: num[:3]
Out[12]: [1, 2, 3]
```

上述是索引的特殊用法，[0:]表示从第 0 个索引开始，直到最后一个元素。[:3]表示从第一个元素开始，直到第 3 个索引。

```
In [13]: num[-1:]
Out[13]: [5]

In [14]: num[:-1]
Out[14]: [1, 2, 3, 4]

In [15]: num[-2:-1]
Out[15]: [4]

In [16]: num[-3:-1]
Out[16]: [3, 4]
```

负数当然也有特殊用法。[-1:]表示从最后一个元素开始，因为它已经是最后一个元素了，所以只返回它本身。[:-1]表示从第一个元素开始到最后一个元素。num[-2:-1]和 num[-3:-1]大同小异。

数组的增删查

我们已经了解数组的基本概念，不过仍旧停留在查找，它不涉及数据的变化。工作中，更多需要操纵数组，对数组的元素进行添加，删除，更改。

数组通过 insert 函数插入，函数的第一个参数表示插入的索引位置，第二个表示插入的值。

```
In [17]: num.insert(1,9)

In [18]: num
Out[18]: [1, 9, 2, 3, 4, 5]
```

另外一种方式是 append，直接在数组末尾添加上元素。它在之后讲到迭代和循环时应用较多。

```
In [19]: num.append(6)
         num
Out[19]: [1, 9, 2, 3, 4, 5, 6]
```

如果要删除特定位置的元素，用 pop 函数。如果函数没有选择数值，默认删除最后一个元素，如果有，则删除数值对应索引的元素。

```
In [20]: num.pop(1)
         num
Out[20]: [1, 2, 3, 4, 5, 6]

In [21]: num.pop()
Out[21]: 6

In [22]: num
Out[22]: [1, 2, 3, 4, 5]
```

更改元素不需要用到函数，直接选取元素重新赋值即可。

```
In [23]: num[0] = 0
         num
Out[23]: [0, 2, 3, 4, 5]
```

到这里，数组增删改查已经讲完，但这只是一维数组，一维数组之上还有多维数组。如果现在有一份数据是关于学生信息，一共有三个学生，要求包含学生的姓名，年龄，和性别，应该怎么用数组表示呢？

有两种思路，一种是用三个一维数组分别表示学生的姓名，年龄和性别。

```
In [24]: name = ['qinlu','lulu','qinqin']
         sex = ['male','female','male']
         age = [18,19,20]

In [25]: print(name[0],sex[0],age[0])
         qinlu male 18
```

学生属性被拆分成多个数组，利用索引来表示其信息，这里的索引有些类似 SQL 的主键，通过索引查找到信息。但是这种方法并不直观，实际应用会比较麻烦，更好的方法是表示成多维数组。

```
In [26]: student = [
             ['qinlu','male',18],
             ['lulu','female',19],
             ['qinqin','male',20]
         ]

In [27]: student[0]
Out[27]: ['qinlu', 'male', 18]
```

所谓多维数组，是数组内再嵌套数组，图中表示的是一个宽度为 3，高度为 3 的二维数组。此时 student[0]返回的是数组而不是单一值。这种方法将学生信息合并在一起，比第一个案例更容易使用。

如果想选择第一个学生的性别，应该怎么办呢？很简单，后面再加一个索引即可。

```
In [28]: student[0][1]
Out[28]: 'male'
```

现在尝试快速创建一个多维数组。

```
In [29]: row = [0]*3
         row
Out[29]: [0, 0, 0]

In [30]: martix = [row] *4

In [31]: martix
Out[31]: [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

[0]*3 将快速生成 3 个元素值为 0 的数组，这是一种快捷操作，而[row]*4 则将其扩展成二维数据，因为是 4，所以是 3*4 的结构。

这里有一个注意点，当我们想更改多维数组中的某一个元素而不是数组时，这种方式会错误。

```
In [32]: martix[1][0] = 1

In [33]: martix
Out[33]: [[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]]
```

按照正常的想法，martix[1][0]将会改变第二个数组中的第一个值为 1，但是结果是所有数组的第一个值都变成 1。这是因为在 matrix = [row] * 4 操作中，只是创建 3 个指向 row 的引用，可以简单理解成四个数组是一体的。一旦其中一个改变，所有的都会变。

比较稳妥的方式是直接定义多维数组，或者用循环间接定义。多维数组是一个挺重要的概念，它也能直接表示成矩阵，是后续很多算法和分析的基础（不过在 pandas 中，它是另外一种形式了）。

元组

tuple 叫做元组，它和数组非常相似，不过用圆括号表示。但是它最大的特点是不能修改。

```
In [34]: tuple = (1,2,3)
         tuple
Out[34]: (1, 2, 3)
```

当我们想要修改时就会报错。

```
In [36]: tuple.append(1)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-36-da2c0107bb55> in <module>()
----> 1 tuple.append(1)

AttributeError: 'tuple' object has no attribute 'append'
```

而选择和数组没有差异。

```
In [39]: tuple[0]
Out[39]: 1

In [40]: tuple[-1]
Out[40]: 3

In [41]: tuple[0:2]
Out[41]: (1, 2)
```

元组可以作为简化版的数组，因为它不可更改的特性，很多时候可以作为常量使用，防止被篡改。这样会更安全。

字典

字典 dict 全称 dictionary，以键值对 key-value 的形式存储。所谓键值，就是将 key 作为索引存储。用大括号表示。

```
In [43]: dict = {'qinlu':18,'lulu':19,'qinqin':20}

In [44]: dict
Out[44]: {'lulu': 19, 'qinlu': 18, 'qinqin': 20}
```

图中的'qinlu'是 key，18 是 value 值。key 是唯一的，value 可以对应各种数据类型。key-value 的原理不妨想象成查找字典，拼音是 key，对应的文字是 value（当然字典的拼音不唯一）。

字典和数组的差异在于，因为字典以 key 的形式存储和查找，所以它的查询速度非常快，毕竟翻字典的时候你只要知道拼音就能快速定位了。对 dict 数据结构，10 个 key 和 10 万个 key 在查找对应的 value 时速度没有太大差别。

这种查找方式的缺点是占用内存大。数组则相反，查找速度随着元素的增加逐渐下降，这个过程想象成程序在一页页的翻一本没有拼音的字典，直到找到内容。数组的优点是占用的内存空间小。

所以数组和字典的优缺点相反，dict 是空间换时间，list 是时间换空间，这是编程中一个比较重要的概念。实际中，数据分析师的工作不太涉及工程化，选用数组或者字典没有太严苛的限制。

细心的读者可能已经发现，字典定义时我的输入顺序是 qinlu，lulu，qinqin，而打印出来是 lulu，qinlu，qinqin，顺序变了。这是因为定义时 key 的顺序和放在内存的 key 顺序没有关系，key-value 通过 hash 算法互相确定，甚至不同 Python 版本的哈希算法也不同。这一点应用中要避免出错。

既然字典通过 key-value 对匹配查找，那么它自然不能不用数组的数值索引，它只能通过 key 值。

```
In [45]: dict['qinlu']
Out[45]: 18
```

如果 key 不存在，会报错。通过 in 方法，可以返回 True 或 False，避免报错。

```
In [47]: 'qinlu' in dict
Out[47]: True
```

dict 和 list 一样，直接通过赋值更改 value。

```
In [48]: dict['qinlu'] = 17

In [49]: dict['qinlu']
Out[49]: 17
```

能不能更改 key 的名字？不能，key 一旦确定，就无法再修改，好比字典定好后，你能修改字的拼音么？

dict 中删除 key 和 list 一样，通过 pop 函数。增加 key 则是直接赋予一个新的键值对。

```
In [50]: dict.pop('lulu')
         dict
Out[50]: {'qinlu': 17, 'qinqin': 20}

In [51]: dict['luqin'] = 18

In [52]: dict
Out[52]: {'luqin': 18, 'qinlu': 17, 'qinqin': 20}
```

dict 的 keys 和 values 两个函数直接输出所有的 key 值和 value 值。如果要转换成数组，则再外面嵌套一个 list 函数。

```
In [54]: dict.keys()
Out[54]: dict_keys(['luqin', 'qinqin', 'qinlu'])

In [55]: dict.values()
Out[55]: dict_values([18, 20, 17])
```

items 函数，将 key-value 对变成 tuple 形式，以数组的方式输出。

```
In [56]: dict.items()
Out[56]: dict_items([('luqin', 18), ('qinqin', 20), ('qinlu', 17)])

In [57]: list(dict.items())
Out[57]: [('luqin', 18), ('qinqin', 20), ('qinlu', 17)]
```

字典可以通过嵌套应用更复杂的数据格式，和 NoSQL 与 JSON 差不多。

```
In [59]: dict = {
             1:{'name':'qinlu',
                'age':18},
             2:{'name':'lulu',
                'age':19}
         }

In [61]: dict[1]['name']

Out[61]: 'qinlu'
```

基础的数据类型差不多了，更多函数应用大家可以网上自行查阅文档，这块掌握了，在数据清洗过程中将会非常高效，尤其是读取 Excel 数据时。当然不要求滚瓜烂熟，因为后面将学习更加强大的 Numpy 和 Pandas。