Lecture 1　Hello world

金融科技协会　2018 年 9 月 26 日

一　编程环境搭建

1.1　什么是 Anaconda?

图 1: Logo

Anaconda 是一个开源的 Python 发行版本，可以简单理解为：

$$Anaconda = Python + conda + a\ lot\ of\ data\ science\ packages + tools\ like\ Jupyter\ Notebook$$

conda 是一个包管理器和环境管理器 (可以理解为五金店，买工具的地方)：在 Python 的使用，特别是数据处理和分析中，常常用到许多第三方包 (类似工具)，而使用 conda 可以方便地帮助你安装、更新、卸载这些第三方包 (因此类似于五金店)。虽然 Python 官网上的 Python 附带 pip(官方的包管理器)，但 conda 会更加贴心地提示一些包之间的依赖关系，避免许多麻烦。另外，conda 可以进行环境管理：例如 A 项目使用了 Python2，而 B 项目又要使用 Python3，一台电脑同时安装两个版本 Python 会非常混乱，而 conda 可以帮你为每个项目建立一个 Python 环境，自己选择在此环境中使用哪个版本的 Python，各环境彼此独立，互不干扰。

Anaconda 自带 150 多个科学计算包及其依赖项，如 Numpy、Pandas、Scipy 等，并集成了 Spyder、Jupyter Notebook 等工具。

1.2　如何安装 Anaconda?

Anaconda 的下载地址为：https://www.anaconda.com/download/，请根据自己的电脑类型，选择 Python3.6 版本进行下载。且安装过程中选择“添加环境变量”。

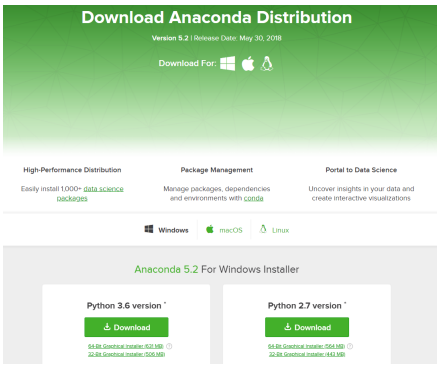

图 2: 下载页面

注：如果计算机上已经安装了 Python，Anaconda 的安装不会造成任何影响——随后程序默认使用 Anaconda 中附带的 Python。

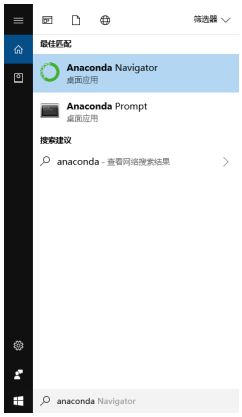

图 3: 安装完成

Anaconda Prompt 是终端界面，需要输入命令进行操作；Anaconda Navigator 是 GUI(图形化) 界面，直接使用鼠标进行点击操作即可。因此，我们仅使用 Prompt 操作 conda 进行包管理，其他操作均使用 Navigator。

图 4: Anaconda Prompt

1.3 如何进行包管理?

使用 Anaconda Prompt 进行包管理，需要记住一些常用命令：

- conda list：查看安装的所有的包；

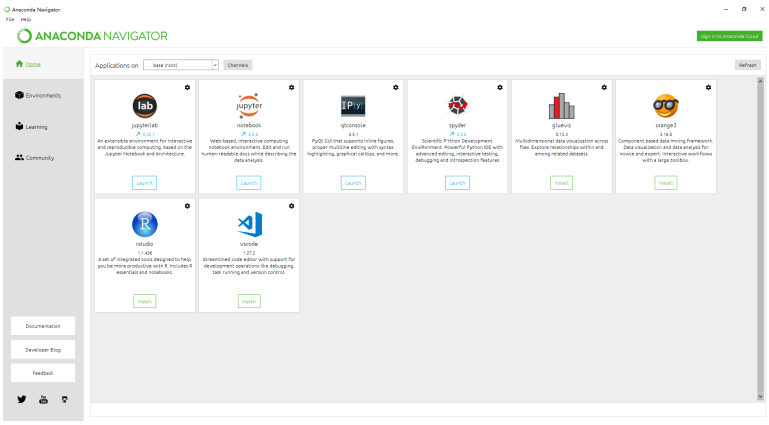

图 5: Anaconda Navigator

- conda upgrade –all：将所有的包更新到最新版本；
- conda install package_name：安装包；
- conda remove package_name：卸载包；
- conda update package_name：更新包；

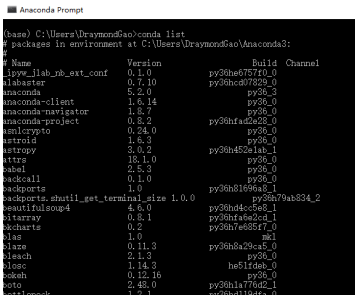

图 6: conda list

1.4 如何进行环境管理?

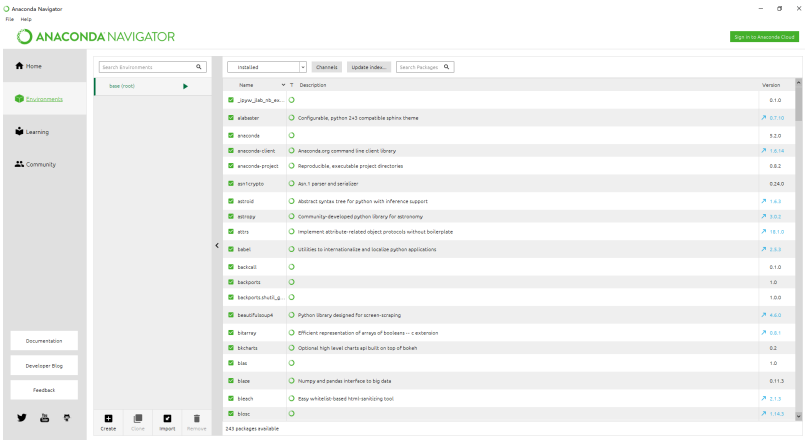

图 7: 环境管理——environments

使用 Anaconda Navigator 进行环境管理。若需要为新项目创建环境，则点击左下角的 *Create*，跳出如下提示框：

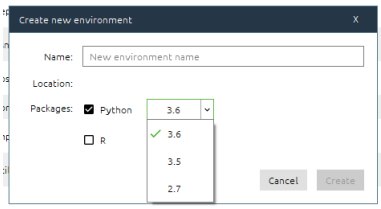

图 8: 可以为环境选择 python 版本

1.5 使用 Jupyter Notebook 的正确姿势

https://www.zhihu.com/question/46309360/answer/254638807

二 变量

2.1 变量

计算机编程中的变量 (Variable) 与数学中的变量含义类似，用于保存特定的“值（Value）”，以便程序在其后多次使用。

2.1.1 赋值

赋值是指将变量值和变量名相连接的过程。使用等号给变量赋值，等号左边是变量名，等号右边是存储在变量中的变量值。赋值完成后，变量值可通过变量名访问。例如，$age = 32$ 表示将整数 32 赋给变量 age，$name =' Sam'$ 表示将字符串 Sam 赋给变量 $name$。注意，由于 Python 是动态语言，因此 Python 中的变量赋值不需要类型声明，其类型由 Python 自动推断。赋值是一个从右往左的过程，$a = b = c = 4$ 表示，将整数 4 赋变量 c，再将变量 c 的值赋给变量 b，再将变量 b 的值赋给变量 a。

每个变量在使用前都必须赋值，否则会导致错误，即 NameError：

```
>>> a = 10
>>> a + b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
>>>
```

图 9: 使用变量 b 前未赋值，导致 NameError

连续为同一变量赋值，则后面的变量值将会覆盖前面的变量值：

```
>>> a = 32          # 将32赋给变量a
>>> b = '三十二'    # 将字符串'三十二'赋给变量b
>>> a = b           # 将b的值，也就是字符串'三十二'赋给了
a，现在a的值是'三十二'，而32被覆盖掉了
>>> a
'三十二'
```

图 10: 变量值的覆盖

多变量同时赋值 (序列赋值)：

```
>>> a,b,c = 1,2,'hello'
>>> a
1
>>> b
2
>>> c
'hello'
```

图 11: 序列赋值

2.1.2 变量名

Python 对变量名是大小写敏感的 (Case-sensitive)，即例如 *spam* 与 *Spam* 就是两个不同的变量。为变量取名需要遵循下列硬性规则:

- 变量名只能包含字母、数字、下划线；
- 变量名不能以数字开头；
- Python 保留字不能被用作变量名。

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

图 12: 常见的保留字

除上述三条强制性规则外，Python 还有一些约定俗成的命名惯例。这些惯例并非强制要求，但遵循它会使代码整洁优雅，可读性更高：

- 常以小写字母开头；
- 驼峰规则：$myCatName$；
- 下划线规则：my_cat_name。

三 基本数据类型

3.1 数值类型

Python 有两种数值类型：整型 *int* 和浮点型 *float*，可以简单区别为整数 (无小数点的数) 和小数 (带小数点的数)。例如：1 是整数，而 1.0 是浮点数 (只要有小数点就是 *float*)。

3.2 布尔类型

布尔类型 (boolean) 又叫做布尔逻辑值，只有两种取值：*True* 和 *False*，含义可按字面理解。一个布尔变量可以代表 *True* 和 *False* 值中的一个。例如，$lightsOn = True$ 将 *True* 赋值给变量 *lightsOn*。在计算机内部，Python 使用 1 来表示 *True*，使用 0 来表示 *False*：

```
>>> int(True)
1
>>> int(False)
0
>>> True == 1
True
>>> False == 0
True
```

图 13: 布尔类型

3.3 字符串

字符串是 Python 序列类型的一种，为不可变序列。

3.3.1 表示

字符串可以用单引号、双引号、三引号表示。单引号和双引号没有什么差别，三引号主要用于界定多行文本，如下，注意每行末尾的回车 (换行符 $\backslash n$) 的体现：

```
>>> """This is
... a big
... test."""
'This is \na big \ntest.'
>>>
```

图 14: 三引号界定多行文本 (注意换行符反斜杠 $\backslash n$)

3.3.2 转义字符

假如你想输出带有引号的字符串，下列做法是错误的：$print(”Sam\ said,”I\ love\ you!””)$，因为当第一个双引号存在时，Python 会自动认为第二个双引号是结尾，上述句子会发生混乱。因此，Python 使用特殊的符号表示特殊的字符——转义字符——由反斜杠 \ 和紧跟其后的字母或字符构成。常用的转义字符有：

转义字符	名称
$\backslash n$	换行符
$\backslash t$	制表符
$\backslash\backslash$	反斜杠
$\backslash'$	单引号
$\backslash”$	双引号

3.3.3 基本操作

加号 +：表示合并。

乘号 $*$：表示重复。

$len()$：得到字符串长度。例如，$Hello$ 由 5 个字符组成，因此 $len('Hello')$ 返回 5。

```
>>> a,b = 'Hello', 'World'
>>> a + b
'HelloWorld'
>>> a * 2 + b * 3
'HelloHelloWorldWorldWorld'
>>> len(a)
5
>>> a[0] + a[len(a)-1]
'Ho'
>>> 'H' in a
True
>>> 'h' in a
False
>>> 'He' in a
True
>>> 'he' in a
False
```

图 15: 字符串基本操作

$X[J]$：索引，得到目标元素 (字符串里的单个字符)。索引从 0 开始，即第一个元素索引为 0，第 n 个元素索引为 $n-1$；

in 和 $not\ in$：判断元素是否在其中，对于字符串来说，判断一个字符串在不在另一个字符串中 (也就是，判断一个字符串是不是另一个字符串的子字符串)。

3.4 列表

列表 (List) 是一个**任意类型的对象**的**位置相关**的**有序**集合。这里有两个关键词，第一个是任意类型，列表中的元素没有固定类型的约束，允许类型不相同，这也是列表和数组的最重要区别（数组这种数据结构，里面的元素类型必须相同）；第二个是位置相关，列表是有序的，每个元素都有自己的位置信息，也就是索引，同样从 0 开始。

3.4.1 表示

列表用中括号表示：将元素用中括号括起，其间逗号连接，即形成列表，如下：

```
>>> A = [1,2,3.7,'Hello',[0,0,0]]
>>> A
[1, 2, 3.7, 'Hello', [0, 0, 0]]
```

图 16: 创建列表

3.4.2 操作

加号 $+$：按照顺序，合并多个列表。

乘号 $*$：表示重复。

$len()$：返回列表内元素的个数。

索引：和字符串一样，第一个是 0，最后一个是 $n-1$；同样支持反向索引，倒数第一个元素索引是 -1，倒数第二个是 -2，以此类推。

```
>>> A = [1,2,'Hello']
>>> A * 3
[1, 2, 'Hello', 1, 2, 'Hello', 1, 2, 'Hello']
>>> B = [3.7,9.7]
>>> A + B
[1, 2, 'Hello', 3.7, 9.7]
>>> len(A)
3
>>> A[0]
1
>>> A[-1]
'Hello'
```

图 17: 列表基本操作 (与字符串类似)

append()：在列表的尾部插入一项。

insert(*index*, *value*)：在任意位置插入元素。

clear()：清除列表中的全部元素，使其成为空的 list。

```
>>> A
[1, 2, 'Hello']
>>> A.append(8)
>>> A
[1, 2, 'Hello', 8]
>>> A.insert(1, 'World')
>>> A
[1, 'World', 2, 'Hello', 8]
>>> A.clear()
>>> A
[]
```

图 18: 列表部分方法

四 运算符

在 Python 中像 +、− 这样的符号，称为运算符；由运算符连接起来的式子称为表达式；操作数指被运算符操作的值，例如 34 + 1 中运算符 + 有两个操作数，分别为 34 和 1。

4.1 算术运算符

变量 a 为 100，变量 b 为 5：

运算符	描述	举例
+	加，将两个对象相加；或表示正数，可省略	a+b 输出结果 105
−	减，将两个对象相减；或表示负数，不可省略	a-b 输出结果 95
*	乘，将两个数相乘；或表示重复	a*b 输出结果 500
/	除，将两个数相除复	a/b 输出结果 20
%	取余，返回除法的余数	a%b 输出结果 0
**	幂，即乘方	a**b 输出结果 10000000000
//	整除，返回除法的整数部分	5//3 输出结果 1

4.2 比较运算符

变量 a 为 100，变量 b 为 5：

运算符	描述	举例
==	等于，比较 a 是否相等 b	a==b 输出结果 False
!=	不等于，比较 a 是否不相等 b	a!=b 输出结果 True
>	大于，返回 a 是否大于 b	a>b 输出结果 True
<	小于，返回 a 是否小于 b	a<b 输出结果 False
>=	大于等于，返回 a 是否大于等于 b	a>=b 输出结果 True
<=	小于等于，返回 a 是否小于等于 b	a<=b 输出结果 False

4.3 赋值运算符

除普通赋值外，我们还会经常遇到变量的当前值被使用、修改然后重新赋值给同一变量的情况。例如，count = count + 1，表示给变量 count 加 1(具体而言，是给 count 所代表的变量值加 1，再把得到的新值赋给变量 count)。Python 中可以使用增强型赋值运算符简写上述过程：count += 1，该表达式等价于 count = count + 1。

运算符	描述	举例
=	普通赋值	c=a+b 表示将 a+b 的结果赋给变量 c
+=	加法赋值	c+=a 等价于 c=c+a
-=	减法赋值	c-=a 等价于 c=c-a
*=	乘法赋值	c*=a 等价于 c=c*a
/=	除法赋值	c/=a 等价于 c=c/a
%=	取余赋值	c%=a 等价于 c=c%a
**=	幂赋值	c**=a 等价于 c=c**a
//=	取整赋值	c//=a 等价于 c=c//a

4.4 逻辑运算符

运算符	描述	举例
and	与	*a and b*，a 与 b 中只要有一个为 False，结果就为 False
or	或	*a or b*，a 与 b 中只要有一个为 True，结果就为 True
not	非	取相反的布尔值；a 为 True，则 *not a* 为 False

4.5 成员运算符

运算符	描述	举例
in	在	*a in b*，如果 a 在序列 b 中，返回 True
not in	不在	*a not in b*，如果 a 不在序列 b 中，返回 True

4.6 运算符优先级

Python 运算符优先级和数学中算术表达式的优先级排序差不多。运算先后顺序大致为：

1. 有括号先算括号里的
2. 指数运算
3. 乘法、除法、余数
4. 加减
5. 比较运算符
6. 赋值运算符
7. 成员运算符
8. 逻辑运算符

五 常用函数

函数在数学中表示输入和输出的映射关系，即输入 x，经过函数 f 后，返回输出 $f(x)$，也可以理解为对输入 x 进行“一番操作”。

Python 中函数的定义与数学中的定义类似，其功能也是对输入进行“一番操作”。例如，*print*() 函数，其中 *print* 是函数名。*print*(”*Hello world*!”) 中，输入值，即字符串 ”*Hello world*!” 被称为参数。*print*() 函数的功能是在控制台输出参数的结果，即“对参数进行一番操作使其显示在控制台上”。

```
1 print("Hello world!")

Hello world!
[Finished in 0.2s]
```

图 19: *print*() 函数

5.1 help()

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

图 20: *print*() 函数的帮助信息

help() 函数用于查看对象的帮助信息。例如，如果你忘了 *print*() 函数的使用方法，可以用 *help*(*print*) 获得 *print*() 函数的帮助信息。

5.2 print()

打印输出。

```
print("Hello,Python!")            # 输出：Hello,Python!
print("Hello","Python!")          # 可以输出多个值，默认空格连接，输出：Hello Python!
print("Hello","Python!",sep='***')# 输出多个值，sep参数指定分隔符，输出：Hello***Python!
```

```
Hello,Python!
Hello Python!
Hello***Python!
[Finished in 0.3s]
```

图 21: *print*() 函数的用法

5.3 type()

返回参数的数据类型。

```
>>> type(100)
<class 'int'>
>>> type(4.3)
<class 'float'>
>>> type('Hello')
<class 'str'>
>>> type(True)
<class 'bool'>
>>> type([1,2,3])
<class 'list'>
```

图 22: *type*() 函数的用法

5.4 input()

函数等待用户在键盘上输入一些文本，并按下回车。该函数返回的结果是用户输入的文本。$myName = input()$ 表示将用户输入的文本赋值给变量 myName。

```
>>> myName = input("What's your name: ")
What's your name: Kobe
>>> myName
'Kobe'
>>>
```

图 23: *input*() 函数的用法

5.5 len()

返回序列的长度。

```
>>> A = 'Hello'
>>> len(A)
5
>>> B = [1,2,3,4]
>>> len(B)
4
```

图 24: *len*() 函数的用法

5.6 str()

将参数转化为字符串类型。

```
>>> A = 10000
>>> type(A)
<class 'int'>
>>> B = str(A)
>>> B
'10000'
>>> type(B)
<class 'str'>
```

图 25: *str*() 函数的用法

5.7 float()

将参数转化为浮点数类型，类似于 *str*() 函数。

5.8 int()

将参数转化为整数类型，类似于 *str*() 函数。

5.9 eval()

eval() 函数用来执行一个字符串表达式，并返回表达式的值。

```
>>> eval('3.85')
3.85
>>> eval('2+2')
4
>>> eval('3**3')
27
```

图 26: *eval*() 函数的用法

5.10 int() 与 eval() 对比

int() 函数和 eval() 函数都可以将整数字符串转化为整数。例如 int(”28”) 会返回 28，eval(”28”) 也会返回 28。但两者稍有区别：

int 函数不能用于转换非整型字符串。例如 int(”2.8”) 会报错，而 eval 可以，eval(”2.8”) 会返回一个 float，即 2.8。从这方面看，eval 功能更加强大。

但 eval 有一个小缺点：不能转换开头有 0 的数字字符串。例如，int(”039”) 会返回 39，但 eval(”039”) 会报错。

六 补充一：继续符号

某些情况下，python 不能确定在多行中哪里是语句的结尾。可以在一行的结尾处放置一个继续符号，即 \，告诉 python 这条语句继续到下一行。

```
sum = 1 + 2 + 3 + 4 + 5 +\
      6 + 7
```

```
sum = 1 + 2 + 3 + 4 + 5 + 6 + 7
```

图 27: 上述两个语句等价

七 补充二：科学计数法

python 中的科学计数法和数学中的一样，一个浮点数可以被写作 $a \times 10^b$。

123456.789 在数学中可以被写作 1.23456789×10^5，在 python 中使用科学计数法则写作 1.23456789E5(或 1.23456789E+5，或 1.23456789e5，或 1.23456789e+5，即 E 可以大写也可以小写)。

0.00012345 在 python 中可以表示为 1.2345E-4。

By Draymond 2.0