

# 第一讲 - Python 概述

张建章

阿里巴巴商学院

杭州师范大学

2023-09



1 jupyter-lab 常见问题与解答

2 Python 下载与安装

3 两种代码执行方式

4 基本数据类型: 整数

5 基本数据类型: 浮点数

6 基本数据类型: 复数

7 基本数据类型: 字符串

8 基本数据类型: 布尔值

9 基本数据类型: 空值

- 10 变量
- 11 操作符
- 12 语句和表达式
- 13 函数
- 14 简单输入与输出
- 15 模块
- 16 简单条件判断语句
- 17 简单循环语句
- 18 写程序注意事项

## (1) jupyter-lab 启动后无法成功运行

- ① 在使用 jupyter-lab 的过程中，确保 cmd 黑框框处于打开状态；
- ② 如果 cmd 黑框框显示 **Bad File Descriptor** 错误，类似下图

```
To access the notebook, open this file in a browser:
  file:///C:/Users/%E5%B8%85%E5%B8%85%E9%A3%9E%E7%8C%AA/AppData/Roaming/j
Or copy and paste one of these URLs:
  http://localhost:8888/?token=47cf2aaa44780278c4e644e8c277c5088e44a5cca0
  or http://127.0.0.1:8888/?token=47cf2aaa44780278c4e644e8c277c5088e44a5cca0
[I 18:56:16.832 NotebookApp] 302 GET / (::1) 0.000000ms
[W 18:56:44.320 NotebookApp] 404 GET /nbextensions/widgets/notebook/js/extension-
00ms referer=http://localhost:8888/notebooks/TFPractise/Untitled.ipynb
Bad file descriptor (C:\projects\libzmq\src\epoll.cpp:100)
Bad file descriptor (C:\projects\libzmq\src\epoll.cpp:100)
```

解决办法：关闭当前 cmd 黑框框，重新打开 cmd 黑框框，在确保网络连通的情况下，依次执行如下两条命令 `pip uninstall pyzmq`；  
`pip install pyzmq==19.0.2 --user`，两条命令成功运行后（运行时没有出现 Error 信息），重新启动 jupyter-lab 即可。

## (2) jupyter-lab 的浏览器界面需要输入 token

请复制你 cmd 黑框框中的 http 开头的网址 (两个网址中的任意一个, 类似下图) 到浏览器打开。

```
To access the notebook, open this file in a browser:
  file:///C:/Users/Administrator/AppData/Roaming/jupyter/runtime/nbserver-58048-open.html
Or copy and paste one of these URLs:
  http://localhost:8888/?token=adda914e1a9f67dc96d78601ae42e1dae7cb10efa3bc06b3
  or http://127.0.0.1:8888/?token=adda914e1a9f67dc96d78601ae42e1dae7cb10efa3bc06b3
[W 23:49:14.572 LabApp] Could not determine jupyterlab build status without nodejs
[I 23:49:16.360 LabApp] Kernel started: bde3aa9e-ea4a-4fa0-a89c-e46129c63b26
```

## (3) 运行 BMI\_calculation 代码时, 输入身高体重后无法继续计算

确保在输入身高体重信息后, 按 **Enter** 键确认, 因为 **input** 函数在接受键盘输入后, 需要用户确认输入, 以继续运行后续程序代码。

## (4) 明明在 jupyter-lab 里写了代码, 却在本机上找不到

请在启动 jupyter-lab 后, 进入到桌面 **Desktop**, 然后新建 **Notebook**, 重命名为有意义的英文名字, 再写代码, 写代码过程中, 一定要多按保存键, 快捷键为 **Ctrl + S**。

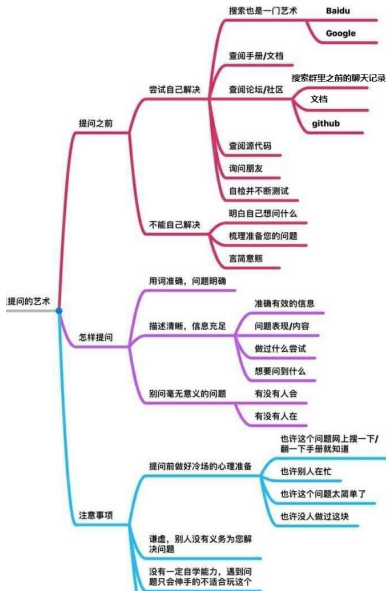
## (5) 课程配套代码的打开与运行

**本地：**①将课程网站的代码下载到本机，建议下载到桌面，便于查找；②启动 jupyter-lab，从左侧文件导航栏找到本地保存的课程代码，双击后打开；③ 运行选中的当前代码行，即当前选中的 cell，点击形如播放的按钮，运行全部代码，点击形如快进的按钮。

**云端：**①将课程网站的代码下载到本机，建议下载到桌面，便于查找；② 打开魔搭在线 jupyter 环境，左侧文件导航栏上方，点击向上的箭头，上传课程代码到云端；③启动 jupyter-lab，从左侧文件导航栏找到本地保存的课程代码，双击后打开；④ 运行选中的当前代码行，即当前选中的 cell，点击形如播放的按钮，运行全部代码，点击形如快进的按钮。

**注意：**一定要自己练习课程提供的代码，切忌只看代码，而不动手写代码和运行自己写的代码。

## (6) 提问的艺术



安装 Python 3.X 并启动: 本课程安装的 Anaconda 3.X 已内含 Python 3.X, 启动 Jupyter-lab 即启动 Python;



Python 采用编译/解释混合方式: 先编译成字节码, 再解释执行。



## 交互式

```
[1]: # 计算BMI指数

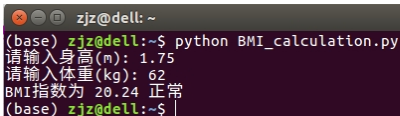
[2]: Height = float(input("请输入身高(m): "))
      请输入身高(m): 1.75

[3]: Weight = float(input("请输入体重(kg): "))
      请输入体重(kg): 62

[4]: BMI = round(Weight/Height**2, 2)

[5]: if BMI>=23.9:
      print("BMI指数为", BMI, "体质偏重")
      elif BMI<=18.5:
          print("BMI指数为", BMI, "体质偏轻")
      else:
          print("BMI指数为", BMI, "正常")
      BMI指数为 20.24 正常
```

## 脚本式



```
zjz@dell: ~
(base) zjz@dell:~$ python BMI_calculation.py
请输入身高(m): 1.75
请输入体重(kg): 62
BMI指数为 20.24 正常
(base) zjz@dell:~$ |
```

## Python 支持多种进制类型

二进制 (Binary, 0 ~ 1), 以 `0b` 或者 `0B` 为前缀;

八进制 (Octal, 0 ~ 7), 以 `0o` 或者 `0O` 为前缀;

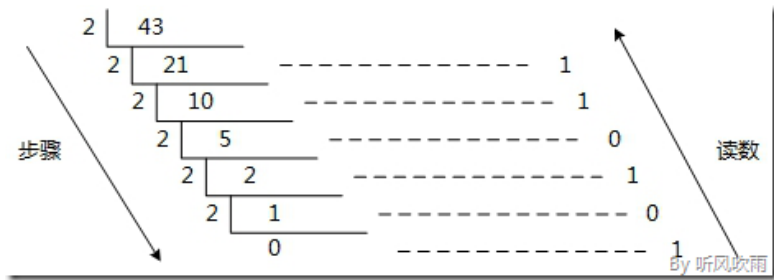
十进制 (Decimal, 0 ~ 9), Python 默认使用的进制, 不需要输入前缀;

十六进制 (Hexadecimal, 0 ~ 9, A ~ F), 以 `0x` 或者 `0X` 为前缀。

**Python 3.7.X+** 可以表示任意大小整数

## 十进制到其他进制 (n) 的转换

**除 n 取余法:** 即每次将整数部分除以 n，余数为该位权上的数，这个步骤一直持续下去，直到商为 0 为止，读数时，从最后一个余数读起。下图为十进制数 43 转化为二进制数 101011 的计算示例。

图 1:  $(43)_D = (101011)_B$

## 其他进制 (n) 到十进制的转换

以二进制为例：二进制数从低位到高位 (从右向左) 计算，第 0 位的权值是 2 的 0 次方，第 1 位的权值是 2 的 1 次方，第 2 位的权值是 2 的 2 次方，依次递增下去，把最后的结果相加的值即为十进制的数值。

八进制、十六进制转十进制的方法与二进制转十进制类似。

二进制数 101011 转换为十进制数，如下：

$$\begin{aligned} 101011 &\rightarrow 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 \\ &= 1 + 2 + 0 + 8 + 0 + 32 \\ &= 43 \end{aligned}$$

## Python 中的进制转换函数

通过内置函数 `bin`, `oct`, `hex` 实现 10 进制转换为其他进制:

```
bin(43) # 0b101011  
oct(796) # 0o1434  
hex(796) # 0x31c
```

通过内置函数 `int` 实现其他进制转十进制:

```
int('101011', 2) # 43  
int('53', 8) # 43  
int('2B', 16) # 43
```

普通计数法和科学计数法 (用 E 或 e 表示底数 10)

```
0.1233333445 # 0.1233333445  
1E-3 # 0.001
```

存在不定尾数, 有些浮点数无法精确表达

```
0.1 + 0.2 # 0.30000000000000004  
0.6 + 1.2 # 1.7999999999999998
```

形如  $z = a + bi$  ( $a$ 、 $b$  均为实数) 的数称为**复数**。其中,  $a$  称为实部,  $b$  称为虚部,  $i$  称为虚数单位。

- 实数可以被认为是虚部为零的复数, 实数  $a$  等价于复数  $a + 0i$ ;
- 实部为零且虚部不为零的复数也被称作**纯虚数**, 如,  $2i$ ;
- 实部不为零且虚部也不为零的复数也被称作**非纯虚数**, 如  $3 + 2i$ 。

**Python 中表示复数要注意:** ① 用字母  $j$  来表示虚数单位  $i$ ; ② 虚部为 1 时, 1 不可以省略。

```
3+2j # (3+2j)
1 + 1j # (1+1j)
```

字符串在解释器中通常高亮显示:

```
'Great Company, 好公司, gute Firma, 良い会社, Хорошая компания, شركة جيدة'
```

```
'Great Company, 好公司, gute Firma, 良い会社, Хорошая компания, شركة جيدة'
```

Python 中使用字符串需注意:

- 字符串写在引号 (单引号、双引号、三引号) 内;
- 字符串可以是空的 (一对引号中什么字符也没有);
- 字符串中包含引号时: 使用**转义字符**, 或者, 字符串边界使用的引号与字符串中的引号不是同一类型 (如, 双引号中可以直接输入单引号作为字符串的一部分)。



布尔值有两个: True (1, 真), False (0, 假), 布尔值可以直接参与运算, True 相当于 1, False 相当于 0。

```
True == 1 # True
```

```
True - 3 # -2
```

```
False + True # 1
```

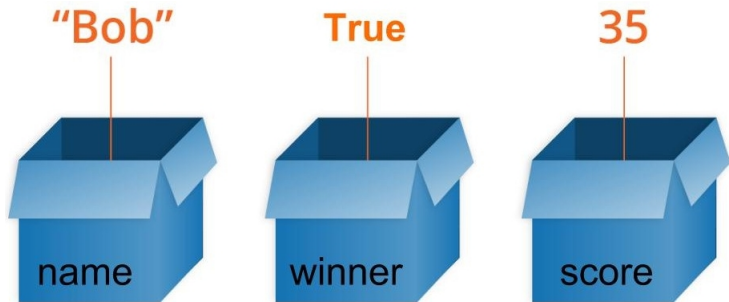
```
1 + true - false # Error, 必须首字母大写
```

空值用 `None` 表示，不能直接参与运算。

```
None + 3 # Error
```

```
False == None # False
```

## 变量的创建




一个数据在计算机内需要一个对应的内存空间，每个内存空间存在一个地址，通过地址程序可以访问内存中的数据，变量名与变量地址绑定，可以通过变量名来访问数据。Python 内置函数 `id` 返回变量的内存地址，用整数表示。

`university_name = 'HZNU'`，`university_name` 是变量名，`HZNU` 是变量值。

## 变量的命名规则

- 只能由字母，数字和下划线组成；
- 不能以数字开头；
- 不能与 `python` 关键字重复；
- 大小写敏感；
- 必须要有意义 (你自己的名字有意义，变量名也要有意义哦)。

## Python 关键字



```
# Python program that prints the complete list of keywords
```

```
import keyword
```

```
print(keyword.kwlist)
```

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

## 运算操作符

Operator	Meaning	Example
<b>+</b>	Addition	$4 + 7 \longrightarrow 11$
<b>-</b>	Subtraction	$12 - 5 \longrightarrow 7$
<b>*</b>	Multiplication	$6 * 6 \longrightarrow 36$
<b>/</b>	Division	$30 / 5 \longrightarrow 6$
<b>%</b>	Modulus	$10 \% 4 \longrightarrow 2$
<b>//</b>	Quotient	$18 // 5 \longrightarrow 3$
<b>**</b>	Exponent	$3 ** 5 \longrightarrow 243$

## 比较操作符

Operators	Meaning	Example	Result
<	Less than	$5 < 2$	False
>	Greater than	$5 > 2$	True
<=	Less than or equal to	$5 \leq 2$	False
>=	Greater than or equal to	$5 \geq 2$	True
==	Equal to	$5 == 2$	False
!=	Not equal to	$5 != 2$	True

## 逻辑操作符

Operator	Meaning	Example	Result
and	Logical and	$(5 < 2)$ and $(5 > 3)$	False
or	Logical or	$(5 < 2)$ or $(5 > 3)$	True
not	Logical not	not $(5 < 2)$	True

在 Python 中，非布尔值也可以与逻辑运算符一起使用：

- 任何非零数字为 `True`；
- 零为 `False`；
- 空字符串为 `False`；
- 非空字符串为 `True`。

当使用非布尔值的逻辑运算符时，Python 返回的是最后评估的值，

```
print(5 and 3) # Output: 3
```



## 身份和成员关系运算符

**身份运算符 `is`**：如果 `is` 两边的变量指向相同的数据对象 (即内存地址相同) 则返回 `True`，否则返回 `False`，可以使用函数 `id` 查看某个变量指向的数据对象所在的内存地址。

**成员关系运算符 `in`**：如果变量或值存在序列中，则返回 `True`，否则返回 `False`。

**等于运算符 `==`**：如果两边的值相同，则返回 `True`，否则返回 `False`。

前两个运算符均可以与 `not` 结合使用，表示否定，`is not`，`not in`。

**`is` 与 `==` 的区别**：双胞胎两个人长得一模一样 (`==`)，但他 (她) 俩不是同一个人 (`is`)。

## 运算符优先级

Precedence	Operator Sign	Operator Name
Highest	**	Exponentiation
	+X, -X, ~X	Unary positive, unary negative, bitwise negation
	*, /, //, %	Multiplication, division, floor, division, modulus
	+, -	Addition, subtraction
	<<, >>	Left-shift, right-shift
	&	Bitwise AND
	^	Bitwise XOR
		Bitwise OR
	==, !=, <, <=, >, >=, is, is not	Comparison, Identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR

## 运算符优先级

先执行优先级高的运算，优先级相同，则从左向右执行（左结合性），为了确保代码的可读性和正确性，在涉及多个运算符的复杂表达式中使用括号来明确指定操作的顺序：

```
x = True
y = False
z = True
# 输出: True, 因为 "not y" 优先于 "and" 和 "or"
print(x and not y or z)
```

```
a = 5
b = 3
c = 8
# 输出: False, 因为 "a < b" 和 "b < c" 的优先级高于 "and"
print(a < b and b < c)
```

### 语句

- 表达某事
- 结果是一个 python 执行动作
- 如，赋值语句 `x = 1`

### 表达式

- 做某事
- 结果是一个值
- 如，逻辑表达式 `x != 1`

**注意：**① 交互解释器 (如 jupyter-lab, Python Shell) 会把所有表达式的值输出；② 语句改变了事物，但没有返回值，也不会有输出 (上面的赋值语句改变了变量 `x` 的值，执行后没有输出，上面的逻辑表达式输出一个布尔值)。

## 算术表达式和赋值语句

**赋值语句：**给变量赋值的语句。

常用的增强的赋值运算符由运算操作符和等号组合而成 (两个符号中间啥也没有哦)，如 `+=`，`/=`，`%=` 等，`x %= 5` 等价于 `x = x % 5`。

**算术表达式：**包含各种算数运算符的计算表达式。

```
# 使用等号进行赋值
```

```
x = 5
```

```
# 使用增强的赋值语句进行赋值
```

```
x += 5 # 等价于 x = x + 5
```

```
# 算术表达式
```

```
2 + 3 * 5 ** 2 % 4 # (2 + ((3 * (5 ** 2)) % 4))
```

**函数：**只有在被调用时才会执行的一个代码块，可以将数据以参数形式传递给函数，函数也可以返回数据作为结果，也可以什么都不返回。

函数调用是一种表达式，如 `print("Hello Kitty")` 就是在调用 Python 的内置函数 `print`，传递的参数为字符串 `"Hello Kitty"`

```
# 定义一个函数
def print_fan_name(fan_name):
    print("Hello Everybody, I am {0}, a fan of Jay Chou
    ↪ (周杰伦)!".format(fan_name))
# 调用函数
fan_name = 'zjzhang'
print_fan_name(fan_name)
```

## 常用内置函数

- `bool` 表示转成布尔值；
- `complex` 表示转成复数，可接收形如 `'1+2j'` 的字符串作为参数；
- `float`，`int` 分别表示转成浮点数或整数；
- `str` 表示转成字符串；
- `chr` 表示 ASCII 值或 Unicode 值转字符；
- `ord` 表示字符转 ASCII 值或者 Unicode 值；
- 点我查阅全部内置函数

ASCII (American Standard Code for Information Interchange, 美国信息交换标准代码) 是基于拉丁字母的一套电脑编码系统。它主要用于显示现代英语，而其扩展版本可以部分支持其他西欧语言 (更多信息点我查看)。Unicode 整理、编码了世界上大部分的文字系统，使得电脑可以用更为简单的方式来呈现和处理文字。

Python 中最常用的输入和输出函数分别为 `input` 和 `print`。

`input` 函数接收任意键盘输入作为参数，并将其转化为字符串，注意，键盘输入后要记得回车确认。

`print` 函数接收任意对象作为参数，并将其打印输出在屏幕，其 `end` 参数是可选的，只接收字符串值，表示打印的结束标志符。

```
# 无论你键盘输入任何内容，input函数都会将其转换为字符串，
```

```
↪ 然后赋值给变量x
```

```
x = input('你输入点东西呗：')
```

```
# end参数默认值为\n，表示换行
```

```
print('你瞅啥') # 你瞅啥
```

```
print('瞅你咋地', end = '!!!') # 瞅你咋地!!!
```



Python 中一个模块对应一个 `.py` 文件，导入模块的语法为 `import` 模块名，亦可以访问模块中的变量或者函数，语句为 `from` 模块名 `import` 函数名/变量名

```
import math
```

```
# 访问模块中的变量-方式1
```

```
math.pi # 3.141592653589793
```

```
# 访问模块中的函数-方式1
```

```
math.sin(0.5*math.pi) # 1.0
```

```
from math import pi, sin
```

```
# 访问模块中的变量-方式2
```

```
pi # 3.141592653589793
```

```
# 访问模块中的函数-方式2
```

```
sin(0.5*math.pi) # 1.0
```

下面尝试用 `sympy` 包中的模块求解高数中的微积分计算题，下面代码分别计算函数  $e^{x^2}$  的导数、极限  $\lim_{x \rightarrow 0} \frac{1}{x}$ 、积分  $\int_0^\infty e^{-x} dx$ 、双重积分  $\int_{-\infty}^\infty \int_{-\infty}^\infty e^{-x^2-y^2} dx dy$ 。

```
# 导入sympy包中的全部模块
from sympy import *

# 定义变量
x, y = symbols("x y")

diff(cos(x), x) # 求导数

limit(1/x, x, 0) # 求极限

integrate(exp(-x), (x, 0, oo)) # 求积分

integrate(exp(-x**2 - y**2), (x, -oo, oo), (y, -oo, oo)) # 求积分
```

不要使用 `sympy` 包代替手算做数学习题!!!

根据条件是否成立决定执行哪种操作，如下：

```
score = float(input('你的成绩是: '))  
# 判断成绩是否及格  
if score >= 60 and score <= 100:  
    print('及格')  
else:  
    print('不及格')
```

一定要记得写条件判断语句中的**冒号**，并且要保证代码对齐，即不同层级的代码要**对齐** (即，缩进的空格数量一致，jupyter-lab 等代码编辑器具有自动缩进对齐代码功能)。

## for 循环

**for** 循环可以遍历序列中的每一个元素，并对元素进行操作，如下：

```
for letter in 'Python':    # 第一个实例
    print("当前字母: %s" % letter)

fruits = ['banana', 'apple', 'mango']
for fruit in fruits:      # 第二个实例
    print ('当前水果: %s'% fruit)

print ("Good bye!")
```

一定要记得写循环语句中的**冒号**，并且要保证代码对齐，即不同层级的代码要**对齐** (即，缩进的空格数量一致，jupyter-lab 等代码编辑器具有自动缩进对齐代码功能)。

## while 循环

`while` 后面的条件为真时，执行循环体内的语句，格式如下：

```
a = 1
while a < 10:
    print(a)
    a += 2
```

[点我查看上述循环执行过程动图](#)

一定要记得写循环语句中的**冒号**，并且要保证代码对齐，即不同层级的代码要**对齐** (即，缩进的空格数量一致，jupyter-lab 等代码编辑器具有自动缩进对齐代码功能)。

## 唯手熟尔

- 避免拼错标识符，如变量名，函数，语句等
- 避免使用中文符号，如引号，逗号，括号等
- 引号、括号通常成对使用，如，有左括号也要有右括号，左边有引号，右边引号也别漏；
- 注意书写格式 (冒号，缩进，对齐)

THE END