SI100+ 2024 Python Lecture 2

变量、运算符和表达式

SI100+ 2024 Staff | 2024-08-24

Part.0 在开始之前

前置知识

在开始这节课之前，先完成前两节录播课：计算机基础知识 & 编程语言和 Python 基础

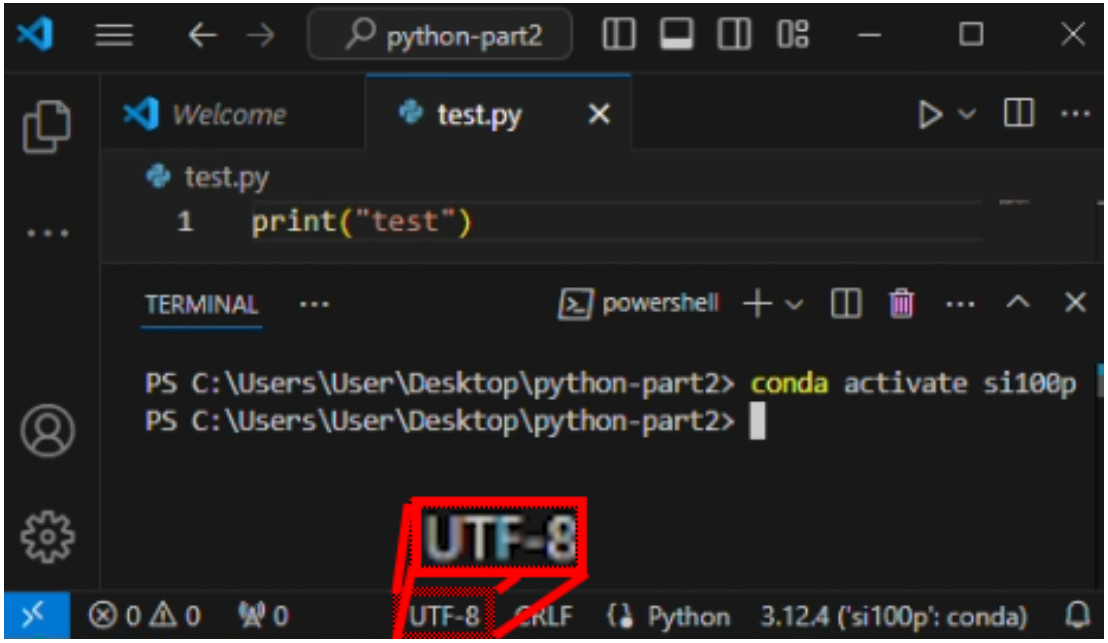
VS Code 基础使用

- **一定要解压缩课程材料压缩包！**
- 在 VS Code 中打开文件夹
- 在 VS Code 中打开终端
- `.ipynb` 和 `.py` 文件都是 Python 文件

文件编码

Python 源代码文件默认使用 UTF-8 编码，与多数操作系统的默认设置一致。如果你的程序出现输出乱码等错误，请首先检查文件编码。

VS Code 中，当前文件的编码显示在窗口下方状态栏右侧，如下图。



Python 的“标准”在哪里？

- **Python 官方文档**：https://docs.python.org/zh-cn/3/
- 菜鸟教程（非官方教程）：https://www.runoob.com/python3/python3-tutorial.html
- **"RTFM"**： Read The *Friendly* Manual

Python 的官方简体中文文档一直在更新，相比于其他语言，非常的友好。

目录

Part.1 Python 基本语法

Python 标识符及其命名规定

标识符 (identifier)：换个说法就是“名字”，唯一地标识一个对象

满足如下规定的一串代码会被 Python 认为是一个标识符：

- 第一个字符必须是字母或下划线 `_` 。
  - 正确示例： `si100p` , `_temp`. 错误示例：`123abc`, `~a`.
- 剩下的部分可由字母、数字及下划线组成。如 `si100p_count`，不可包含空格，如 `cat dog` 会被认为是两个标识符。
- 区分大小写。例如，`si100p` 和 `SI100P` 不是同一个标识符。

演示： Notebook 示例 2.1.1, 2.1.2

Python 标识符及其命名规定

编程中还有所谓的 **命名法 / 命名规则**，也就是“起名习惯”。规范较多，大家自行搜索学习。

（非课程内容）几个例子如下：

- **驼峰命名法**：首字母小写，后面每个单词首字母大写，如 `myName`, `myAge`
- **匈牙利命名法**：变量名之前写明变量类型，如 `strName`, `intAge`
- **下划线命名法**：单词之间用下划线 `_` 分隔，如 `my_name`, `my_age`
- **帕斯卡命名法**：每个单词首字母大写，如 `MyName`, `MyAge`

在 PEP-8 中，Python 社区总结了一套较为统一的命名规范：
https://peps.python.org/pep-0008/

Python 关键字

关键字 (keyword)：对于 Python 来说有特殊含义的词，这些词**不能被用作标识符名称**

Python 内置模块 `keyword` 记录有当前版本的所有关键字。

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',
...
```

演示：Notebook 示例 2.1.3，2.1.4

Python 的缩进

Python 的特色之一是依据代码行的 **缩进 (indentation)** 确定代码块：

```
if 1 + 1 == 2:
    print("basic statement passed!") # 我和 if 语句不在同一个代码块!

a = 10
b = 20
# 上面的两行在同一个代码块
```

当前只需了解缩进这一概念即可，之后的课程中，我们会了解到缩进与代码块在 Python 中的意义。

> VS Code 中，默认按 Tab 在光标处插入缩进，Shift + Tab 移除光标前的缩进。

Python 的注释

注释 (comment)：代码文件中不会被执行的文本

注释虽然不影响程序执行，但会使代码更容易被阅读和理解。

Python 的注释分为**单行**和**多行**注释两种。

Python 的注释

单行注释以井号 # 开头，井号 # **后直到行尾**的内容将被视为注释：

```
# 我是单行注释!
print("我能正常执行!") # 我也是单行注释!
# print("我怎么也变成注释了? 我不会被执行了! ")
```

```
# 这一行被注释了
print("注释的效果不会保持到下一行")
```

演示： Notebook 示例 2.1.5

将某一语句变为注释可快捷地让其不执行，在程序出现错误需要改正时较为实用。

> VS Code 中，默认按 `Ctrl + /` 可以注释或取消注释当前行。

Python 的注释

Python **没有专用的多行注释**，但通常使用多行字符串达成近似效果，即连用三个单引号 `'''` 或双引号 `"""` 括起注释内容：

```
'''
我是一个被用作注释的多行字符串。
下面是一个 print 语句。

空行不会打断多行字符串的效果。

print("这行代码不会被执行")
'''


print("这行代码可以被正常执行")
```

演示： Notebook 示例 2.1.6

Part.2 字面值与赋值语句

字面值

字面值 (literal value)：代码中写明的、即时使用的临时值，例如语句 `x = 1.5` 中的 `1.5`。

字面值涵盖几乎所有基本数据类型，我们将在讲解对应数据类型时作介绍，此处不再赘述。

赋值语句

字面值不适合复杂计算（比如，将当前结果暂存以备稍后使用）。

可将字面值（以及计算结果）**赋值 (assign)** 给一个变量，对应语句称为**赋值语句**。

- 例如 `x = 1.5` 就是一个赋值语句。
  - 意为“令 x 的值为 1.5” / “将 1.5 赋给 x”。
  - 此处的 x 即为变量。
- 所以“变量”到底是什么？

Part.3 变量和基本数据类型

变量

变量 (variable)： 用于存储数据的“容器”。

- 高中数学 “未知数”

每个变量都有一个名字，可通过它来引用存储在变量中的数据。 本课开头所讲的命名规则适用于任何命名，变量命名也不例外。同时，命名应尽可能**有意义并反映实际用途**。

```
age = 25            # 整型数字变量
height = 175.5      # 浮点型数字变量
name = "Alice"      # 字符串变量
is_student = True # 布尔型变量
```

演示： Notebook 示例 2.3.1

> 注：Python 中，变量声明和赋值通常在同一条语句中完成。

基本数据类型

Python 中的基本数据类型用于表示和操作不同种类的数据，主要有以下三类：

- 数字 (Number)
- 字符串 (String)
- 布尔 (Boolean)

数字 (Number)

数字 (Number) 数据类型用于存储数值，其中又包含四种类型的数值。本课程只涉及两种最常用的：

- **整型 (`int`)**：用于表示任意整数，可以是正数、负数或零，例如 `10`, `-3`, `0` 等。
  - 整型数据无小数部分
- **浮点型 (`float`)**：，用于表示带有小数部分的数值，包括正浮点数和负浮点数，例如 `3.14`, `-0.001`, `2.0` 等。
  - 浮点型数据用于表示更精确的数值，常用于科学计算和测量值。
  - **注意**：Python 在涉及除法 `/` 运算时，输出结果总是浮点型。

字符串 (String)

字符串 (str) 是由一系列字符组成的文本数据，外部由引号（单引号或双引号）括起来。例如：`"Hello world!"`, `'Python'` 等。

字符串适合用于存储和操作文本，可以包含字母、数字、符号和空格。

字符串支持加法运算 `+`，效果是将前后字符串相连接。例如，`"SI" + "100P"` 的结果是 `"SI100P"`.

布尔 (Boolean)

布尔型 (`bool`) 数据只有两个值：`True`（真）和 `False`（假）。

布尔型数据通常表示二元状态（如开/关、是/否），常用于条件判断和逻辑运算。

Python 的数据类型 - 演示

Python 内置的 `type()` 能告诉我们变量属于哪个数据类型：

```
>>> x = "text"
>>> type(x) # 变量 x 是字符串类型
<class 'str'>
```

现在让我们实验一下变量赋值语句，并尝试借助变量进行简单数学计算（加 +、减 -、乘 *、除 / 等），如 `x = 8 / 4`.

演示： Notebook 示例 2.3.2

等等.....

```
>>> x = 0.1
>>> y = 0.2
>>> print(x + y)
0.30000000000000004
```

为什么不是 0.3?

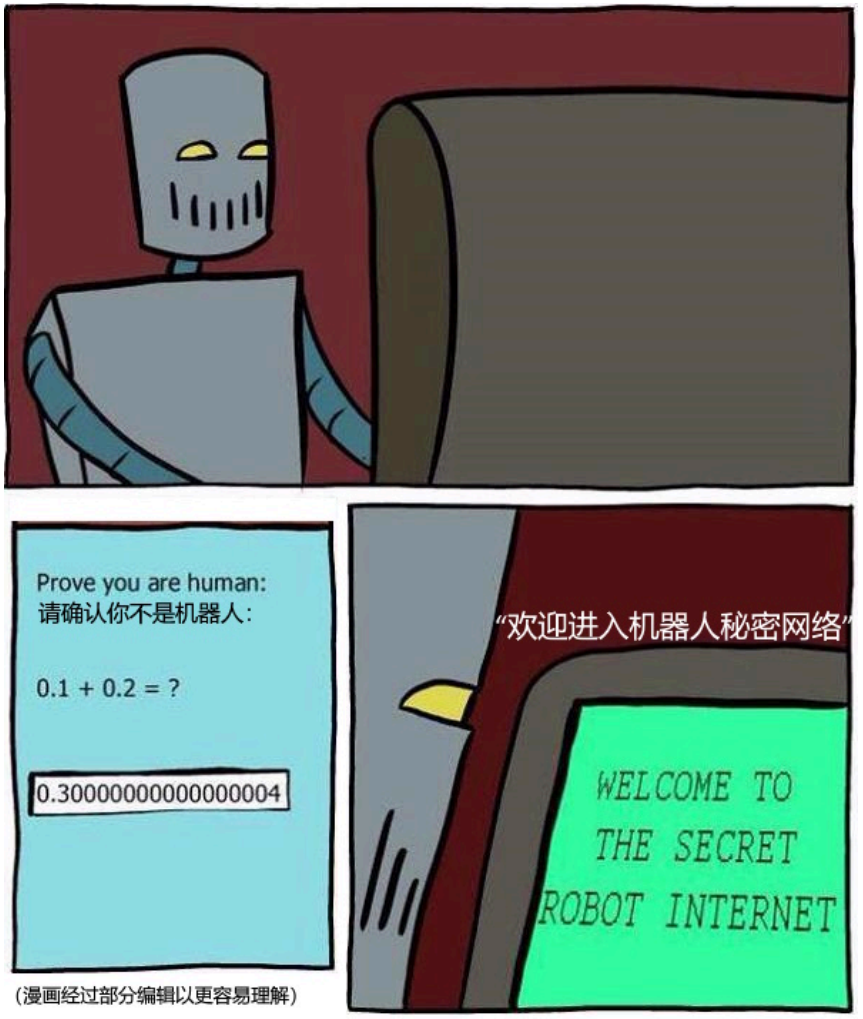
演示： Notebook 示例 2.3.3

浮点数与其误差

浮点数的表示： 浮点数以二进制格式存储，采用科学记数法表示，即 $m \times 2^e$，其中 m 是尾数，e 是指数。 现代计算机大多遵循与上述格式相同的 IEEE 754 标准来表示浮点数。

网站 https://float.exposed/ 对浮点数的存储格式作了直观的视觉演示，有兴趣可以自行查看了解。

meme source



(漫画经过部分编辑以更容易理解)

浮点数与其误差

浮点数误差的来源： 许多十进制浮点数无法精确表示为二进制浮点数（如 0.5 是 2^{-1}但是 0.1 在二进制中是一个无限循环的小数），只能以近似表示存储。因此浮点数的运算结果可能会出现**舍入误差**。

- 0.1 实际存储的值近似于十进制的 0.1000000000000000055511...
- 0.2 实际存储的值近似于十进制的 0.2000000000000000111022...
- 两个值相加，得到的结果近似于十进制的 0.30000000000000000444...
  - 注：浮点数是以二进制形式存储相加，因此把近似的十进制值相加后，结果对不上是正常的。

而 0.3 实际存储的值接近于 0.29999999999999999889... 所以就出现了 "0.1 + 0.2 ≠ 0.3" 的情况。

浮点数与其误差

一般情况下，你可以通过四舍五入保留小数点后的有限位来解决这类问题；如果对计算精度要求较高，最好使用 decimal 等计算模块。

```
from decimal import Decimal

x = Decimal('0.1') # 使用 Decimal 类型存储 0.1
```

演示： Notebook 示例 2.3.4

Part.4 运算符，优先级，输入输出

运算符与优先级

Python 中的 **基本运算符 (operator)** 有：+, -, *, /, //, %, **。

- //（整除）为除法结果舍去小数位（取商）。如 5 // 3 结果为 1.
- %（取余/求模）为取得两数相除后的余数。如 5 % 3 结果为 2.
- ** 为指数运算。如 5 ** 3 即为 5^3。

演示： Notebook 示例 2.4.1

运算符**优先级 (precedence)**： ** > 正负号 (+x, -x) > [*, /, //, %] > [+, -]

计算时，运算符优先计算更深层括号内的，处于同一层级括号则先计算优先级较高的，优先级相同则从左至右计算。

（扩展）Python 运算符优先级表

运算符与优先级：分步示例

粗体 -> 上一步计算结果

高亮 -> 下一步计算对象

$$3 * 3 + 5 \% 3 + 16 ** (1/2)$$

运算符与优先级：分步示例

3 * 3 + 5 % 3 + 16 ** (1/2)

运算符与优先级：分步示例

3 * 3 + 5 % 3 + 16 ** (1/2)

3 * 3 + 5 % 3 + 16 ** **(0.5)**

运算符与优先级：分步示例

3 * 3 + 5 % 3 + 16 ** (1/2)

3 * 3 + 5 % 3 + 16 ** (0.5)

运算符与优先级：分步示例

$$3 * 3 + 5 \% 3 + 16 ** (1/2)$$

$$3 * 3 + 5 \% 3 + 16 ** (0.5)$$

$$3 * 3 + 5 \% 3 + \mathbf{4}$$

运算符与优先级：分步示例

3 * 3 + 5 % 3 + 16 ** (1/2)

3 * 3 + 5 % 3 + 16 ** (0.5)

3 * 3 + 5 % 3 + 4

运算符与优先级：分步示例

$$3 * 3 + 5 \% 3 + 16 ** (1/2)$$

$$3 * 3 + 5 \% 3 + 16 ** (0.5)$$

$$3 * 3 + 5 \% 3 + 4$$

$$\mathbf{9} + 5 \% 3 + 4$$

运算符与优先级：分步示例

$$3 * 3 + 5 \% 3 + 16 ** (1/2)$$

$$3 * 3 + 5 \% 3 + 16 ** (0.5)$$

$$3 * 3 + 5 \% 3 + 4$$

$$9 + 5 \% 3 + 4$$

运算符与优先级：分步示例

$$3 * 3 + 5 \% 3 + 16 ** (1/2)$$

$$3 * 3 + 5 \% 3 + 16 ** (0.5)$$

$$3 * 3 + 5 \% 3 + 4$$

$$9 + 5 \% 3 + 4$$

$$9 + \mathbf{2} + 4$$

运算符与优先级：分步示例

$$3 * 3 + 5 \% 3 + 16 ** (1/2)$$

$$3 * 3 + 5 \% 3 + 16 ** (0.5)$$

$$3 * 3 + 5 \% 3 + 4$$

$$9 + 5 \% 3 + 4$$

$$9 + 2 + 4$$

运算符与优先级：分步示例

$$3 * 3 + 5 \% 3 + 16 ** (1/2)$$

$$3 * 3 + 5 \% 3 + 16 ** (0.5)$$

$$3 * 3 + 5 \% 3 + 4$$

$$9 + 5 \% 3 + 4$$

$$9 + 2 + 4$$

15

运算符与优先级：分步示例

演示： Notebook 示例 2.4.2

```
>>> 3 * 3 + 5 % 3 + 16 ** (1/2)
15.0
>>> type(3 * 3 + 5 % 3 + 16 ** (1/2))
<class 'float'> # 牵涉到除法，输出为浮点型
```

基本输入输出

输入输出 (Input/Output 简称 I/O) 是程序读取和输出数据的基本途径。

有两种输入输出类型：标准 (Standard I/O) 和文件 (File I/O).

标准输入输出的操作目标是命令行/控制台，通常用于交互式环境（REPL，Read-Eval-Print Loop）。其中：

- `input()` 用于从用户输入获取数据。
- `print(x)` 用于在屏幕上打印变量 `x`，结尾默认附加一个换行。
  - `print(x, end="abc")` 可以把结尾的换行符替换为 `abc`。
  - 相似地，`print(x, end="")` 可去掉默认附加的换行。
- 使用 `input(x)` 可在接收输入时先输出变量 `x`，以便提示用户输入。
- `input()` 将一切输入作为字符串 (`str`) 处理。
- 使用 `print(x, y, ...)` 将在一行内依序输出 `x`, `y`, ...（以空格间隔）
  - 例如 `print("非常好SI", 100, "P")` 将输出 `非常好SI 100 P`.

演示： Notebook 示例 2.4.3

基本输入输出

文件输入输出的操作目标是硬盘上的文件。 可以使用 Python 内置的 `open()` 打开文件进行读取或写入操作：

```
f = open("test.txt", "r") # 以读取模式打开 test.txt 文本文件
content = f.read() # 读取文件的全部内容
f.close() # 操作完毕，关闭文件

f = open("test.txt", "w") # 以写入模式打开 test.txt 文本文件
f.write("Hello, World!") # 从文件开头覆盖写入字符串
f.write("This is a text.") # 从上一操作处继续写入
f.close() # 操作完毕，关闭文件
```

`open()` 操作默认不支持中文。 如要操作中文，多数情况下需要用 `codec` 模块在指定编码下读取。

由于时间关系，详细的文件操作不在此展开讲述，可自行阅读文档或查阅资料。

演示： Notebook 示例 2.4.4

示例：A + B 问题

以目前所学的知识，可以编写下面这个程序了：接收用户输入的两个整数，计算并输出两者的和。

例：

```
>>> 输入整数 a:
<<< 3
>>> 输入整数 b:
<<< 2
>>> a + b = 5
```

演示： Notebook 示例 2.4.5，2.4.6

示例：A + B 问题

为什么我写的 A + B 程序，输入 12 和 34 会输出 1234?

`input()` **将一切输入作为字符串 (`str`) 处理**。

字符串支持加法运算 `+`，效果是将前后字符串相连接。

"12" + "34" → "1234"

强制类型转换

对于某一类型 typename 我们可以通过 typename(x) 强制转换数据 x 为 typename 类型。

```
>>> x = "42"    # 字符串（str）变量
>>> x
'42'
>>> type(x)
<class 'str'>
>>> y = int(x) # 将 x 转换为整型（int）赋值给变量 y
>>> y
42
>>> type(y)
<class 'int'>
>>> x           # int(x) 不改变原变量 x 的值
'42'
```

强制类型转换

对于某一类型 typename，我们可以通过 typename(x) 强制转换数据 x 为 typename 类型。

一些典型的用途包括：

- 将数字与其字符串形式互转，如 "4.2"（字符串）和 4.2（数字）。
- 将浮点数小数位移除变为整数，例如 int(5.9) 的结果是 5。

注意：类型转换不会导致被转换的变量发生变化。 比如在上一页的例子中，执行 int(x) 之后，x 仍然是字符串类型，其值也没有改变。

> 备注：类型转换 typename(x) 的本质是“用 x 创建一个新的 typename 类型的变量”，因此并非所有类型都能相互转换。而“创建指定类型的变量”则涉及到“类”等高级的编程概念，目前还不需要你详细理解。

演示： Notebook 示例 2.4.7，2.4.8

Part.5 比较运算符和布尔运算

布尔类型 - Recall

布尔型 (bool) 数据只有两个值：`True`（真）和 `False`（假）。

布尔型数据通常表示二元状态（如开/关、是/否），常用于条件判断和逻辑运算。

比较运算符

比较运算符 (comparison operators) 用于比较两个值，其结果是一个布尔值，代表该比较式是否成立。

- ==：判断相等 **（注意是 2 个等号，不要与赋值运算符 = 混淆）**
- !=：判断不等 (感叹号 ！ 后跟等号 =)
- >：判断大于
- <：判断小于
- >=：判断大于等于（大于号 > 后跟等号 =）
- <=：判断小于等于（小于号 < 后跟等号 =）

演示：Notebook 示例 2.5.1

布尔运算符与布尔运算

布尔运算符 (boolean operators) 也称逻辑运算符 (logical operators)，其对布尔值进行布尔运算（也称逻辑运算）。

- `and`： 逻辑与运算 仅当其左右两侧均为 `True` 时，结果为 `True`；否则为 `False`
- `or`： 逻辑或运算 其左右任一侧为 `True` 时，结果为 `True`；否则为 `False`
- `not`： 逻辑非运算 一元运算符，将其右侧布尔值取反（`True` 变为 `False`， 反之亦然）

变量 A	变量 B	A and B 的结果	A or B 的结果	not A 的结果
True	True	True	True	False
True	False	False	True	(同上)
False	True	False	True	True
False	False	False	False	(同上)

演示： Notebook 示例 2.5.2

布尔运算符的短路求值

- 在 `A and B and C and D and ...` 这样的表达式中，如果 A 为 `False`，则无论 B, C, D ... 的值如何，整个表达式必然为 `False`，因此后续的表达式就没有必要计算下去了。
- 类似的，在 `A or B or C or D or ...` 中，如果 A 为 `True`，则整个表达式必然为 `True` 。

Python 的布尔运算符就能够这样“偷懒”。

布尔运算符 and 和 or 是**短路运算符（short-circuit operators）**：其参数从左至右求值，一旦可以确定结果，就不再继续求值。

演示： Notebook 示例 2.5.3，2.5.4

逻辑表达式

逻辑表达式 (logical expression) 用于判断多个条件是否满足某种逻辑关系，并返回布尔值作为结果（`True` 或 `False`）。

其通常由比较运算符和布尔运算符组成。例如：`age >= 18 and has_ticket` 等。

演示： Notebook 示例 2.5.5

常见运算符优先级

类型转换 > ** > 正负号 (+x, -x) > [*, /, //, %] > [+, -] > 比较运算符 (==, !=, <, <=, >, >=) > 布尔运算符 (and, or, not, ...) > ...

Takeaway Message

- 我们发布的资料一定要**解压缩后再使用**！
- VS Code 基础操作
- Python 官方文档是最权威的 Python 参考资料
  - 建立起“遇事不决先上网查文档/资料”的习惯
  - **RTFM** = Read The *Friendly* Manual
  - **STFW** = Search The *Friendly* Web
- 标识符起名规则、缩进和注释
  - 善加利用注释，为他人更为自己

Takeaway Message (cont'd)

- 字面值、赋值语句：如何把字面值赋给一个变量
- 变量的概念（类比高中数学的“未知数”）
  - 如何创建（声明&赋值）变量
- 基本数据类型：数字（整型和浮点型）、字符串、布尔
  - 浮点型数字的误差
  - 部分类型之间可强制转换
- 运算符及其优先级
  - 基本计算、比较、布尔（逻辑）运算
    - 短路运算符：and or
- 表达式（一般/逻辑）及化简顺序

Thanks for Listening

Any questions?