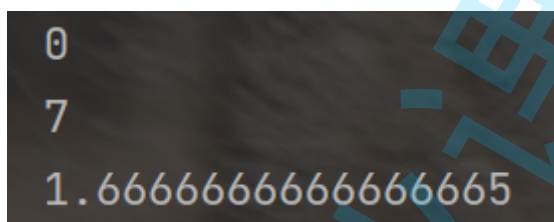


基础语法(1)

常量和表达式

我们可以把 Python 当成一个计算器, 来进行一些算术运算.

```
print(1 + 2 - 3)
print(1 + 2 * 3)
print(1 + 2 / 3)
```



```
0
7
1.6666666666666665
```

注意:

- print 是一个 Python 内置的 **函数**, 这个稍后详细介绍.
- 可以使用 + - * / () 等运算符进行算术运算. 先算乘除, 后算加减.
- 运算符和数字之间, 可以没有空格, 也可以有多个空格. 但是一般习惯上写一个空格(比较美观).

PS: 美观是否重要?

形如 `1 + 2 - 3` 这样是算式, 在编程语言中称为 **表达式**, 算式的运算结果, 称为 **表达式的返回值**

其中 `1`, `2`, `3` 这种称为 **字面值常量**, `+` `-` `*` `/` 这种称为 **运算符** 或者 **操作符**.

注意: 熟悉 C / Java 的同学可能认为, `2 / 3` 结果为 0 (小数部分被截断). 但是在 Python 中得到的结果则是一个小数. 更符合日常使用的直觉.

示例

给定四个分数, `67.5`, `89.0`, `12.9`, `32.2`, 编写代码, 求这四个分数的平均数.

```
print( (67.5 + 89.0 + 12.9 + 32.2) / 4 )
```

变量和类型

变量是什么

有的时候, 我们需要进行的计算可能更复杂一些, 需要把一些计算的中间结果保存起来. 这个时候就需要用到 **变量**.

示例

给定四个分数, 67.5, 89.0, 12.9, 32.2, 编写代码, 求这四个分数的方差.

PS: 方差的计算过程: 取每一项, 减去平均值, 计算平方, 再求和, 最后除以 (项数 - 1)

在这个代码中, 就需要先计算这四个数字的平均值, 然后再计算方差. 这就需要将计算的平均值使用 **变量** 保存起来.

```
avg = (67.5 + 89.0 + 12.9 + 32.2) / 4
total = (67.5 - avg) ** 2 + (89.0 - avg) ** 2 + (12.9 - avg) ** 2 + (32.2 - avg)
** 2
result = total / 3
print(result)
```

注意:

- avg, total, result 均为变量.
- ** 在 Python 中表示乘方运算. ** 2 即为求平方.

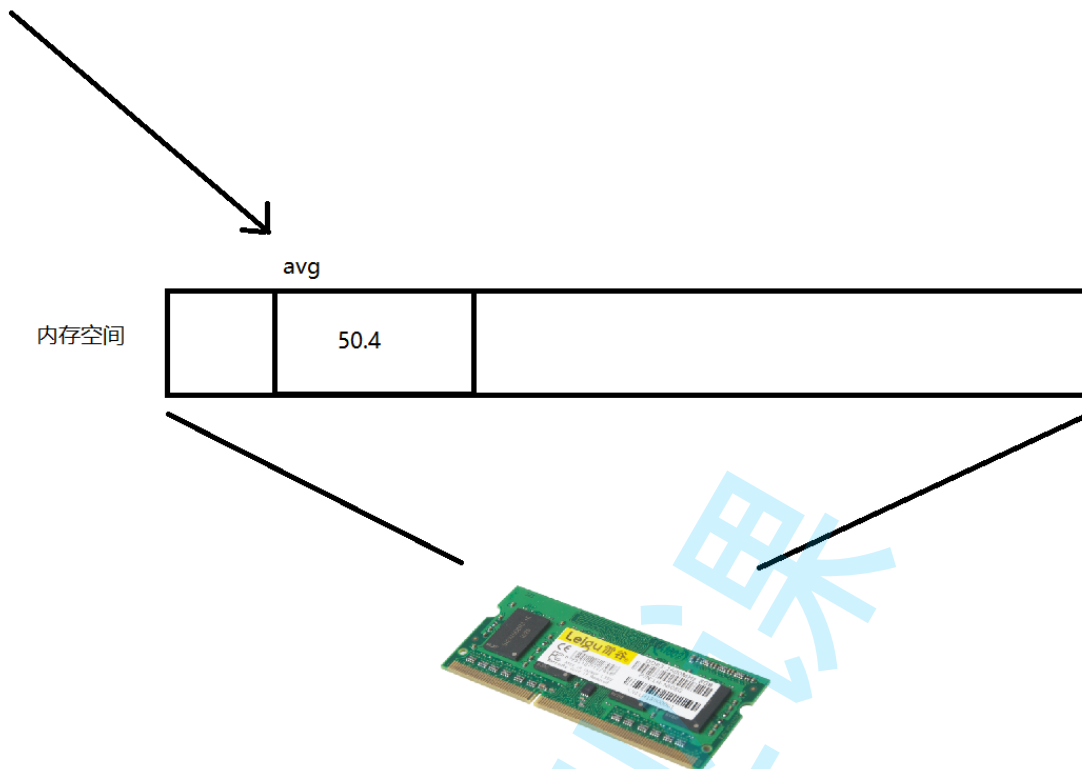
就像计算器中的 M 键功能类似, 通过变量就可以保存计算过程中的中间结果.



只不过, 计算器一般只能保存一个数据, 而在 Python 代码中, 可以创建任意多的变量, 来随心所欲的保存很多很多的数据.

变量可以视为是一块能够容纳数据的空间. 这个空间往往对应到 "内存" 这样的硬件设备上.

```
avg = (67.5 + 89.0 + 12.9 + 32.2) / 4
```



PS: 我们可以把内存想像成是一个宿舍楼, 这个宿舍楼上有很多的房间. 每个房间都可以存放数据.

衡量内存的一个重要指标就是内存空间的大小, 比如我的电脑内存是 16GB. 这个数字越大, 意味着内存的存储空间就越大, 能够存放的数据(变量)就越多.

变量的语法

(1) 定义变量

```
a = 10
```

创建变量的语句非常简单, 其中

- a 为变量名. 当我们创建很多个变量的时候, 就可以用名字来进行区分.
- = 为赋值运算符, 表示把 = 右侧的数据放到 = 左侧的空间中.

注意: 变量的名字要遵守一定规则.

硬性规则(务必遵守)

- 变量名由数字字母下划线构成.
- 数字不能开头.
- 变量名不能和 "关键字" 重复.
- 变量名大小写敏感. num 和 Num 是两个不同的变量名.

软性规则(建议遵守)

- 变量名使用有描述性的单词来表示, 尽量表达出变量的作用.

- 一个变量名可以由多个单词构成, 长一点没关系, 但是含义要清晰.
- 当变量名包含多个单词的时候, 建议使用 "驼峰命名法". 形如 `totalCount`, `personInfo` 这种, 除了首个单词外, 剩余单词首字母大写.

数学上, 变量通常使用 `x`, `y`, `z` 这种简单的英文字母或者拉丁字母表示. 但是在编程中不建议这样使用.

原因是编程中, 一个程序里通常会同时创建出很多个变量. 如果只是使用单个字母表示, 在变量多了的时候, 就很难记住哪个变量是干啥的, 从而给维护程序带来了一定的困难. 因此我们更建议使用带有明确描述性的名字, 来表示变量的用途.

(2) 使用变量

读取变量的值

```
a = 10
print(a)
```

修改变量的值

```
a = 20
print(a)
```

注意: 在 Python 中, 修改变量也是使用 `=` 运算, 看起来和定义变量没有明显区别.

当然, 也可以用一个变量的值赋给另外一个变量.

```
a = 10
b = 20

a = b

print(a)
print(b)
```

变量的类型

变量里面存储的不仅仅是数字, 还可以存储其它种类的数据. 为了区分不同种类的数据, 我们引入了 "类型" 这样的概念.

注意: 和 C++ / Java 等语言不同, Python 变量的类型不需要显式指定, 而是在赋值的时候确定的.

(1) 整数

```
a = 10
print(type(a))
```

```
<class 'int'>
```

PS: type 和 print 类似, 也是 python 内置的函数. 可以使用 type 来查看一个变量的类型.

注意: 和 C++ / Java 等语言不同, Python 的 int 类型变量, 表示的数据范围是没有上限的. 只要内存足够大, 理论上就可以表示无限大小的数据.

(2) 浮点数(小数)

```
a = 0.5
print(type(a))
```

```
<class 'float'>
```

注意: 和 C++ / Java 等语言不同, Python 的小数只有 float 一种类型, 没有 double 类型. 但是实际上 Python 的 float 就相当于 C++ / Java 的 double, 表示双精度浮点数.

PS: 关于单精度浮点数和双精度浮点数的问题, 我们此处不做过多讨论. 大家只要知道, 相比于单精度浮点数, 双精度浮点数占用的内存空间更多, 同时表示的数据精度更高即可(大概精确到小数点后 15 位).

(3) 字符串

```
a = 'hello'
print(type(a))
```

```
<class 'str'>
```

使用 '' 或者 "" 引起来的, 称为 **字符串**. 可以用来表示文本.

注意: 在 Python 中, 单引号构成的字符串和双引号构成的字符串, 没有区别. 'hello' 和 "hello" 是完全等价的.

可以使用 len 函数来获取字符串的长度.

```
a = 'hello'
print(len(a))
```

可以使用 + 针对两个字符串进行拼接.

```
a = 'hello'
b = 'world'
print(a + b)
```

此处是两个字符串相加. 不能拿字符串和整数/浮点数相加.

字符串作为开发中最常用到的数据类型, 支持的操作方式也是非常丰富的. 此处暂时不详细展开.

(4) 布尔

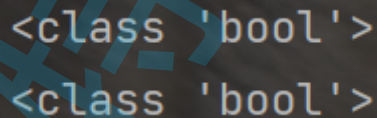
布尔类型是一个特殊的类型, 取值只有两种, True (真) 和 False (假).

PS: 布尔类型也是数学上的一个概念. 我们初中就学过一个概念叫做 "命题", 进一步的就可以判定命题的真假.

例如:

- 汤老湿真帅! (真命题)
- 汤老湿是个妹子 (假命题)

```
a = True
print(type(a))
b = False
print(type(b))
```



```
<class 'bool'>
<class 'bool'>
```

布尔类型在咱们后续进行逻辑判断的时候, 是非常有用的.

(5) 其他

除了上述类型之外, Python 中还有 list, tuple, dict, 自定义类型 等等. 我们后续再介绍.

为什么要有这么多类型?

(1) 类型决定了数据在内存中占据多大空间.

例如 float 类型在内存中占据 8 个字节.

PS: 计算机里面使用二进制来表示数据. 也就是每个位只能表示 0 或者 1.

1 个二进制位, 就称为是一个 "比特", 8 个二进制位, 就称为一个 "字节" (Byte)

一个 float 变量在内存中占据 8 个字节空间, 也就是 64 个二进制位.

我的电脑有 16GB 的内存空间, 也就是一共有 $1024 * 1024 * 1024 * 8$ 这么多的二进制位.

(2) 类型其实约定了能对这个变量做什么样的操作.

例如 int / float 类型的变量, 可以进行 + - * / 等操作

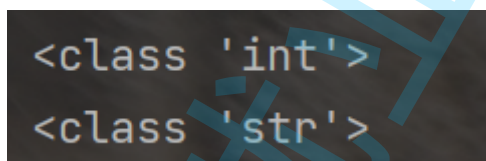
而 str 类型的变量, 只能进行 + (并且行为是字符串拼接), 不能进行 - * / , 但是还能使用 len 等其他操作.

总结: 类型系统其实是在对变量进行 "归类". 相同类型的变量(数据) 往往具有类似的特性和使用规则.

动态类型特性

在 Python 中, 一个变量是什么类型, 是可以在 "程序运行" 过程中发生变化的. 这个特性称为 "动态类型".

```
a = 10
print(type(a))
a = 'hello'
print(type(a))
```



在程序执行过程中, a 的类型刚开始是 int, 后面变成了 str.

C++/Java 这样的语言则不允许这样的操作. 一个变量定义后类型就是固定的了. 这种特性则称为 "静态类型".

动态类型特性是一把双刃剑.

- 对于中小型程序, 可以大大的解约代码量(比如写一段代码就可以同时支持多种类型).
- 对于大型程序, 则提高了模块之间的交互成本. (程序猿 A 提供的代码难以被 B 理解).

注释

注释是什么

注释是一种特殊的代码, 它不会影响到程序的执行, 但是能够起到解释说明的作用, 能够帮助程序猿理解程序代码的执行逻辑.

PS: 写代码是一件比较烧脑的事情, 读代码同样也非常烧脑. 相比于一板一眼的代码, 一些口语化的描述能更好的帮助程序猿理解程序.


```
# 计算 4 个数字 67.5, 89.0, 12.9, 32.2 的方差
avg = (67.5 + 89.0 + 12.9 + 32.2) / 4
total = (67.5 - avg) ** 2 + (89.0 - avg) ** 2 + (12.9 - avg) ** 2 + (32.2 - avg)
** 2
result = total / 3
print(result)
```

形如上述代码, 如果没有注释, 直接阅读, 是不容易 get 到代码的含义是计算方差. 但是通过加了一行注释解释一下, 就让人一目了然了.

PS: 代码的第一目标是容易理解, 第二目标才是执行正确.

写注释不光是为了方便别人来理解, 也是方便三个月之后的自己理解.

一个反例: 早些年医生的手写处方



电话: 139

姓名: 性别: 年龄: 24岁

诊断:

处方:

沙参20g 连翘25g 苏子15g 菊花15g

防风18g 当归20g 川芎2g 山楂30g

半边莲3g 香附15g

鱼腥草18g 乌药18g 三七1g

甘草2g 首乌藤50g

医生: 司药: 2012年 8月 28

注释的语法

Python 中有两种风格的注释.

(1) 注释行

使用 # 开头的行都是注释.

```
# 这是一行注释.
```

(2) 文档字符串

使用三引号引起来的称为 "文档字符串", 也可以视为是一种注释.

- 可以包含多行内容,
- 一般放在 文件/函数/类 的开头.
- """ 或者 ''' 均可 (等价).

```
"""  
这是文档字符串  
这是文档字符串  
"""
```

注释的规范

1. 内容准确: 注释内容要和代码一致, 匹配, 并在代码修改时及时更新.
2. 篇幅合理: 注释既不应该太精简, 也不应该长篇大论.
3. 使用中文: 一般中国公司都要求使用中文写注释, 外企另当别论.
4. 积极向上: 注释中不要包含负能量(例如 领导 SB 等).

输入输出

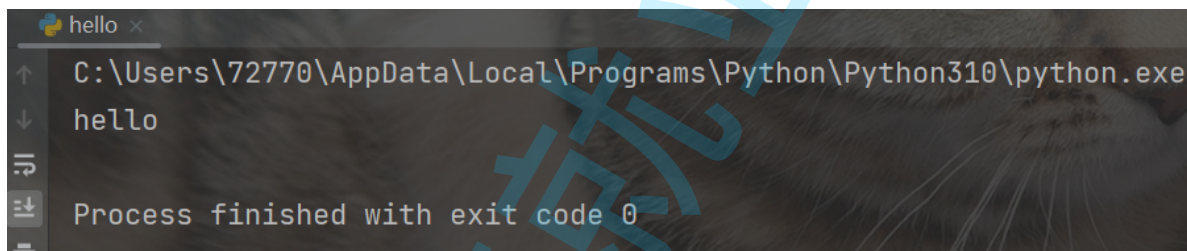
和用户交互

程序需要和用户进行交互.

- 用户把信息传递给程序的过程, 称为 "输入".
- 程序把结果展示给用户的过程, 称为 "输出".

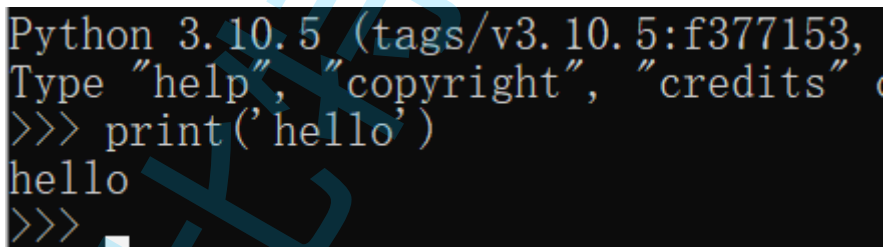
输入输出的最基本的方法就是控制台. 用户通过控制台输入一些字符串, 程序再通过控制台打印出一些字符串.

PyCharm 运行程序, 下方弹出的窗口就可以视为是控制台.



```
hello ×
C:\Users\72770\AppData\Local\Programs\Python\Python310\python.exe
hello
Process finished with exit code 0
```

windows 自带的 cmd 程序, 也可以视为是控制台.



```
Python 3.10.5 (tags/v3.10.5:f377153,
Type "help", "copyright", "credits"
>>> print('hello')
hello
>>>
```

输入输出的最常见方法是图形化界面. 如我们平时用到的 QQ, 浏览器, steam 等, 都不需要用户输入命令, 而只是通过鼠标点击窗口点击按钮的方式来操作.

Python 当然也可以用来开发图形化界面的程序. 但是图形化程序开发本身是一个大话题. 咱们课堂上暂时不做介绍.

通过控制台输出

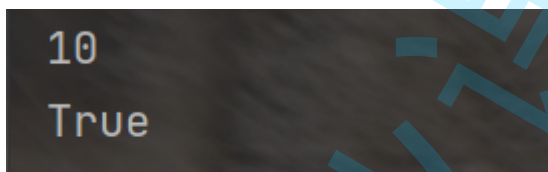
Python 使用 print 函数输出到控制台.

```
print('hello')
```

不仅能输出一个字符串, 还可以输出一个其他类型的变量

```
a = 10
print(a)

b = True
print(b)
```

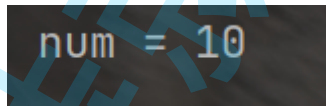


```
10
True
```

更多的时候, 我们希望能够输出的内容是混合了字符串和变量的.

示例: 输出 `num = 10`

```
num = 10
print(f'num = {num}')
```



```
num = 10
```

注意:

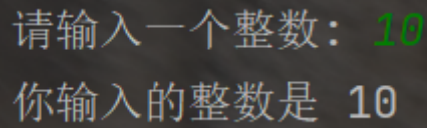
- 使用 f 作为前缀的字符串, 称为 f-string
- 里面可以使用 {} 来内嵌一个其他的变量/表达式.

PS: Python 中还支持其他的格式化字符串的方法, 咱们此处只了解这个最简单的即可. 其他的暂时不做介绍.

通过控制台输入

python 使用 input 函数, 从控制台读取用户的输入.

```
num = 0
num = input('请输入一个整数: ')
print(f'你输入的整数是 {num}')
```



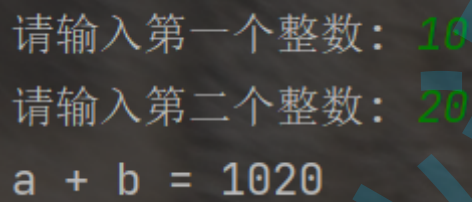
```
请输入一个整数: 10
你输入的整数是 10
```

注意:

- input 的参数相当于一个 "提示信息", 也可以没有.
- input 的返回值就是用户输入的内容. 是字符串类型.

```
a = input('请输入第一个整数: ')
b = input('请输入第二个整数: ')

print(f'a + b = {a + b}')
```



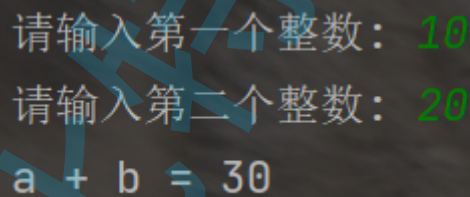
```
请输入第一个整数: 10
请输入第二个整数: 20
a + b = 1020
```

此处的结果是字符串拼接, 不是算术运算. 如果想进行算术运算, 需要先转换类型.

```
a = input('请输入第一个整数: ')
b = input('请输入第二个整数: ')

a = int(a)
b = int(b)

print(f'a + b = {a + b}')
```



```
请输入第一个整数: 10
请输入第二个整数: 20
a + b = 30
```

通过 int() 把变量转成了 int 类型.

类似的, 使用 float(), bool(), str() 等可以完成对应的类型转换.

代码示例: 输入 4 个小数, 求 4 个小数的平均值.


```
a = input('请输入第一个数字: ')
b = input('请输入第二个数字: ')
c = input('请输入第三个数字: ')
d = input('请输入第四个数字: ')

a = float(a)
b = float(b)
c = float(c)
d = float(d)

avg = (a + b + c + d) / 4

print(f'平均值: {avg}')
```

```
请输入第一个数字: 10.2
请输入第二个数字: 20.5
请输入第三个数字: 30.4
请输入第四个数字: 40.5
平均值: 25.35
```

此处为了输入 4 个数字, 执行了四次 input. 如果是读取任意多个数字怎么办呢? 这个时候就需要用到循环了. 后面再介绍.

运算符

算术运算符

像 `+` `-` `*` `/` `%` `**` `//` 这种进行算术运算的运算符, 称为 **算术运算符**

注意1: `/` 中不能用 0 作为除数. 否则会 **抛出异常**

```
print(10 / 0)
```

```
Traceback (most recent call last):
  File "D:\project\classroom_code\python\hello\hello.py", line 1, in <module>
    print(10 / 0)
ZeroDivisionError: division by zero
```

异常 是编程语言中的一种常见机制, 表示程序运行过程中, 出现了一些 "意外情况", 导致程序不能继续往下执行了.

注意2: 整数 / 整数 结果可能是小数. 而不会截断

```
print(1 / 2)
```

注意3: % 不是 "百分数", 而是求余数.

```
print(7 % 2)
```



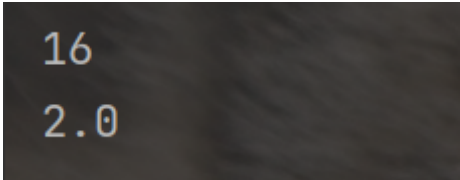
1

关于求余数, 有些同学容易蒙. 其实这个是小学二年级数学就学过的.

7 除以 2, 商是 3, 余数是 1.

注意4: ** 是求乘方. 不光能算整数次方, 还能算小数次方.

```
print(4 ** 2)
print(4 ** 0.5)
```



16
2.0

注意5: // 是取整除法(也叫地板除). 整数除以整数, 结果还是整数(舍弃小数部分, 并向下取整. 不是四舍五入)

```
print(7 // 2)
print(-7 // 2)
```



3
-4

关系运算符

像 < <= > >= == != 这一系列的运算符称为 **关系运算符**, 它们是在比较操作数之间的关系.

其中

- <= 是 "小于等于"
- >= 是 "大于等于"
- == 是 "等于"
- != 是 "不等于"

(1) 如果关系符合, 则表达式返回 True. 如果关系不符合, 则表达式返回 False

```
a = 10
b = 20

print(a < b)
print(a <= b)
print(a > b)
print(a >= b)
print(a == b)
print(a != b)
```

```
True
True
False
False
False
True
```

(2) 关系运算符不光针对整数/浮点数进行比较, 还能针对字符串进行比较.

```
a = 'hello'
b = 'world'

print(a < b)
print(a <= b)
print(a > b)
print(a >= b)
print(a == b)
print(a != b)
```

```
True
True
False
False
False
True
```

注意:

- 直接使用 == 或者 != 即可对字符串内容判定相等. (这一点和 C / Java 不同).
- 字符串比较大小, 规则是 "字典序"

关于字典序:

想象一个英文词典, 上面的单词都是按照字母顺序排列. 如果首个字母相同, 就比较第二个字母. (就比如著名单词 abandon).

我们认为一个单词在词典上越靠前, 就越小. 越靠后, 就越大.

(3) 对于浮点数来说, 不要使用 `==` 判定相等.

```
print(0.1 + 0.2 == 0.3)
```

False

注意: 浮点数在计算机中的表示并不是精确的! 在计算过程中, 就容易出现非常小的误差.

```
print(0.1)
print(0.2)
print(0.3)
print(0.1 + 0.2)
```

```
0.1
0.2
0.3
0.30000000000000004
```

可以看到, `0.1 + 0.2` 的结果并非是 `0.3`, 而是带了个小尾巴. 虽然这个尾巴非常小了, 但是 `==` 是锱铢必较的, 仍然会导致 `==` 的结果为 `False`.

不止是 Python 如此, 主流编程语言都是如此. 这个是 IEEE754 标准规定的浮点数格式所引入的问题. 此处我们不做过多讨论.

正确的比较方式: 不再严格比较相等了, 而是判定差值小于允许的误差范围.

```
a = 0.1 + 0.2
b = 0.3
print(-0.000001 < (a - b) < 0.000001)
```

实际工程实践中, 误差在所难免, 只要保证误差在合理范围内即可.

逻辑运算符

像 `and` `or` `not` 这一系列的运算符称为 **逻辑运算符**.

- `and` 并且. 两侧操作数均为 `True`, 最终结果为 `True`. 否则为 `False`. (一假则假)
- `or` 或者. 两侧操作数均为 `False`, 最终结果为 `False`. 否则为 `True`. (一真则真)
- `not` 逻辑取反. 操作数本身为 `True`, 则返回 `False`. 本身为 `False`, 则返回 `True`.

此处说的 "并且" 和 "或者", 就是我们日常生活中使用的 "并且" 和 "或者".

想象一下未来丈母娘问你要彩礼, 什么叫做 "有房并且有车", 什么叫做 "有房或者有车".

```
a = 10
b = 20
c = 30

print(a < b and b < c)
print(a < b and b > c)

print(a > b or b > c)
print(a < b or b > c)

print(not a < b)
print(not a > b)
```

```
True
False
False
True
False
True
```

一种特殊写法

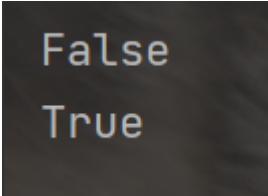
`a < b and b < c` 这个操作等价于 `a < b < c`. 这个设定和大部分编程语言都不相同.

关于短路求值

和其他编程语言类似, Python 也存在短路求值的规则.

- 对于 `and`, 如果左侧表达式为 `False`, 则整体一定为 `False`, 右侧表达式不再执行.
- 对于 `or`, 如果左侧表达式为 `True`, 则整体一定为 `True`, 右侧表达式不再执行.

```
print(10 > 20 and 10 / 0 == 1)
print(10 < 20 or 10 / 0 == 1)
```



False
True

上述代码没有抛出异常, 说明右侧的除以 0 操作没有真正执行.

赋值运算符

(1) `=` 的使用

`=` 表示赋值. 这个我们已经用过很多次了. 注意和 `==` 区分.

`=` 除了基本的用法之外, 还可以同时针对多个变量进行赋值.

链式赋值

```
a = b = 10
```

多元赋值

```
a, b = 10, 20
```

代码实例: 交换两个变量

基础写法

```
a = 10  
b = 20  
  
tmp = a  
a = b  
b = tmp
```

基于多元赋值

```
a = 10  
b = 20  
  
a, b = b, a
```

(2) 复合赋值运算符

Python 还有一些 **复合赋值运算符**. 例如 `+=` `-=` `*=` `/=` `%=`

其中 `a += 1` 等价于 `a = a + 1`. 其他复合赋值运算符也是同理.

```
a = 10
a = a + 1
print(a)

b = 10
b += 1
print(b)
```

注意: 像 C++ / Java 中, 存在 `++ --` 这样的自增/自减运算符. Python 中则不支持这种运算. 如果需要使用, 则直接使用 `+= 1` 或者 `-= 1`

`++ --` 最大的问题就是容易分不清前置和后置的区别. 这一点 Python 语法在设计的时候就进行了规避, 避免出现这种不直观, 并且容易混淆的语法.

其他...

除了上述之外, Python 中还有一些运算符, 比如 **身份运算符** (`is`, `is not`), **成员运算符** (`in`, `not in`), **位运算符** (`&` | `~` `^` `<<` `>>`) 等.

此处咱们暂时不介绍.

总结

本章节中我们学习了 Python 中的最基础的语法部分

- 常量
- 变量
- 类型
 - 整数
 - 浮点数
 - 字符串
 - 布尔值
- 注释
- 输入输出
- 运算符
 - 算术运算符
 - 关系运算符
 - 逻辑运算符
 - 赋值运算符

当前我们的代码还只能进行一些简单的算术运算. 下个章节中我们将学习 Python 中的逻辑判断, 然后我们就可以写稍微复杂一点的程序了.

自测练习

(1) [多选] 以下关于变量之间加法运算的说法, 正确的是:

- A. Python 中的字符串之间够能相加.
- B. Python 中的字符串可以和整数相加.
- C. Python 中的整数可以和浮点数相加.
- D. Python 中的整数可以和布尔值相加.

(2) [单选] 以下关于类型的说法, 正确的是:

- A. Python 中既有字符串类型, 也有字符类型.
- B. Python 中既有 float, 也有 double.
- C. Python 中既有 int, 也有 long
- D. Python 中的整数表示范围无上限.

(3) [单选] 以下 Python 代码, 合法的是

- A. `int a = 10`
- B. `a = 10;`
- C. `a = true`
- D. `a = 'aaa' + 10`