

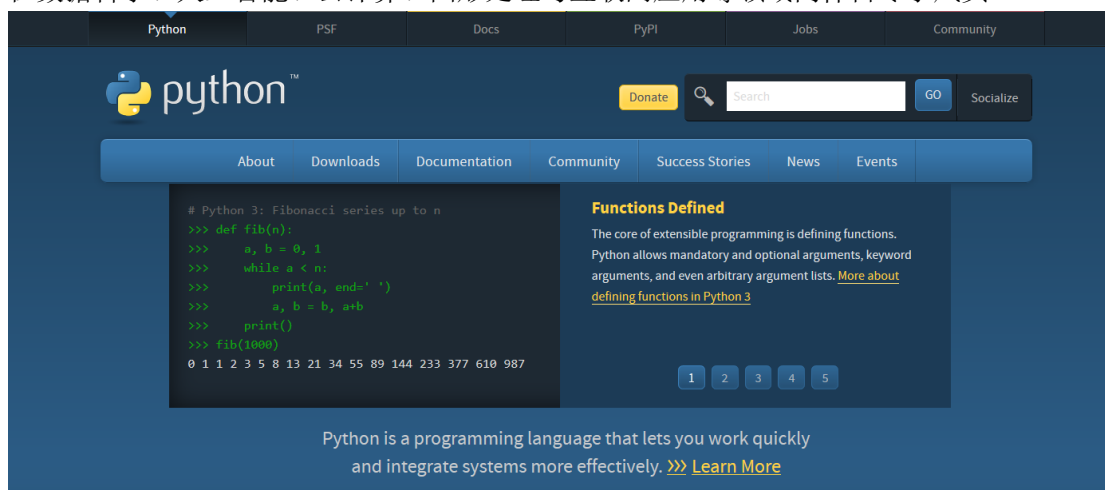
Python 介绍及基础语法

1.1 Python 介绍

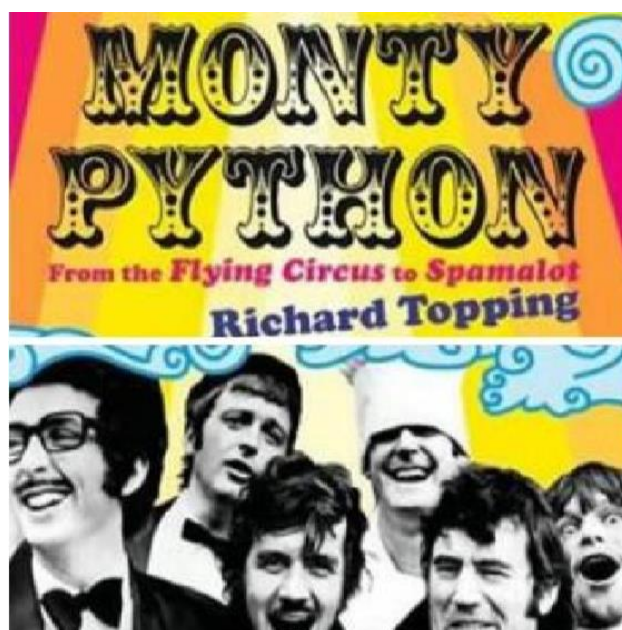
1.1.1 简介

Python 是当今世界最流行的程序语言之一，它通俗易懂可读性强且拥有优秀法结构。Python 是一种解释型、面向对象的语言。由吉多·范罗苏姆(Guido van Rossum)于 1989 年发明，1991 年正式公布。Python 官网：www.python.org，如图下。如果你是一个编程初学者，你会发现 Python 编程并没有那么枯燥，甚至可以体会到 Python 的优雅与简洁之美，希望本期课程将会是你 IT 生涯的启蒙导师。

Python 突出的简洁性、易读性和扩展性，使得 Python 应用于科学研究的机构日益增多，这里也包括一些全球顶尖的大学也在采用 Python 教授程序设计课程。除此之外，Python 在数据科学、人工智能、云计算、图形处理与互联网应用等领域同样占尽了风头。



Python 单词是“大蟒蛇”乱意思。但是龟叔不是喜欢蟒蛇才起这个名字，而是正在追剧：英国电视喜剧片《蒙提·派森的飞行马戏团》(Monty Python and the Flying Circus)。



1.1.2 特点:

1. 可读性强!

可读性远比听上去重要的多得多。一个程序会被反复的修改,可读性强意味着让你可以在更短时间内学习和记忆,直接提高生产率。

2. 简洁,简洁,简洁!

研究证明,程序员每天可编写的有效代码数是有限的。完成同样功能只用一半的代码,其实就是提高了一倍的生产率。

3. Python 是由 C 语言开发,但是不再有 C 语言中指针等复杂数据类型!

Python 的简洁性让开发难度和代码幅度大幅降低,开发任务大大简化。程序员再也不需要关注复杂的语法,而是关注任务本身。

4. 入门级语言!

只适合菜鸟?准确的说拉近了高手与初学者之间的距离。Python 简洁的语法结构,学习门槛低,编程极易上手,无论老鸟还是菜鸟都站在同一个起跑线上。

5. 解释性与交互性!

与典型的 Java 编译型语言相比,Python 属于解释型语言。一方面,Python 编写一条程序语句,即可解释执行返回一个结果。当程序出错时更容易跟踪与定位;另一方面:Python 这种交互式模式为人机互动提供了更广阔的可能空间。

6. 优秀的模块化思维!

将代码组织为一个或若干模块,模块组织成为包、甚至库。试想当你编写程序的时候如果已经有针对科学计算、爬虫、数据分析、可视化、机器学习等模块或第三方库可以直接拿来使用,编程效率会极大的提高。

7. 开源软件!

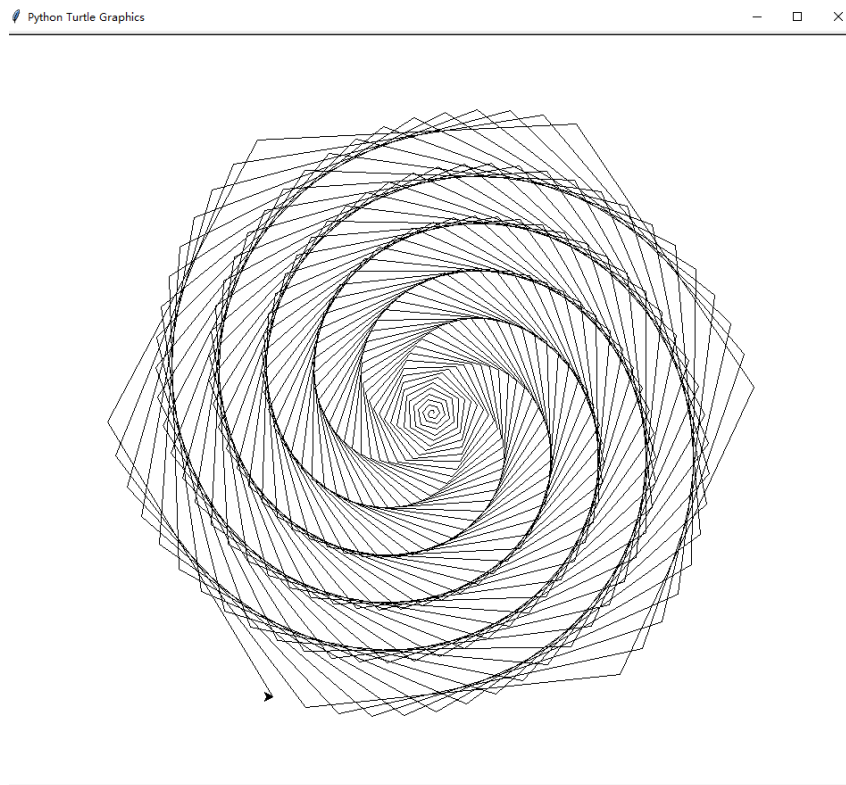
Python 是纯粹的开源语言,源代码遵循 GPL 许可,这些特性使其更受大众欢迎,软件更容易移植到其他的平台,如 Mac、Linux 等,因此 Python 拥有丰富的第三方资源库是不足为奇的。

8. 标准脚本语言!

脚本程序是指只有需要被调用的时候才会被动态的解释执行。Python 允许混合使用 C、Java 与 Python 代码,通过增强扩展性来解决一些特殊的问题,例如 Python 程序中允许调用一段由 Java 编写的程序模块(库),甚至这段 Java 模块可以是保密的。以上这些陈述都充分体现了 Python 的可扩展性和作为脚本语言的动态灵活性。

*GPL 许可协议(GNU General Public License):只要软件中包含有其他 GPL 协议的产品或代码,那么该软件就必须也采用 GPL 许可协议且开源及免费。即复制自由、传播自由,允许以各种形式进行传播,并且收费传播,修改自由。

例子 1.1



完成这样的螺旋线，代码只有几行：

```
1. import turtle
2. t=turtle.Pen()
3. for x in range(0,360):
4.     t.forward(x)
5.     t.left(59)
```

优点：

1. 面向对象
2. 免费和开源
3. 可移植性和跨平台

Python 会被编译成与操作系统相关的二进制代码，然后再解释执行。这种方式和 java 类似，大大提高了执行速度，也实现了跨平台。

4. 丰富的库（丰富的标准库，多种多样的扩展库）
5. 可扩展性。可嵌入到 C 和 C++语言。胶水式语言。

应用范围：

1. 科学计算
2. 人工智能
3. WEB 服务端和大型网站后端。

YouTube、Gmail 等应用基于 python 开发。

4. GUI 开发（图形用户界面开发）
5. 游戏开发
6. 移动设备
7. 嵌入式设备

8. 系统运维

9. 大数据

什么时候不应该用 python

Python 是解释执行。性能较低。

因此，一些影响性能的功能可以使用 C/C++/JAVA/GO（GO 是一种新语言，写起来像 Python，性能像 Java）去开发。不过，也不用太担心，Python 解释器会越来越快。

1.2 IDLE 开发环境使用入门

1.2.1 IDLE 介绍

1. IDLE 是 Python 官方标准开发环境，Python 安装完后同时就安装了 IDLE。
2. IDLE 已经具备了 Python 开发的几乎所有功能（语法智能提示、不同颜色显示不同类型等等），也不需要其他配置，非常适合初学者使用。
3. IDLE 是 Python 标准发行版内置的一个简单小巧的 IDE，包括了交互式命令行、编辑器、调试器等基本组件，足以应付大多数简单应用。
4. IDLE 是用纯 Python 基于 Tkinter 编写，最初级的作者正是 Python 之父 Guido van Rossum。

1.2.2 IDLE 实操

1. 交互模式
启动 IDLE，默认就是进入交互模式。
2. 编写和执行 Python 源文件

1.2.3 快捷键

快捷键	说明
Alt+N Alt+P	查看历史命令上一条、下一条
Ctrl+F6	重启 shell，以前定义的变量全部失效
F1	打开帮助文档
Alt+/ Ctrl + [Ctrl +]	自动补全前面曾经出现过的单词 缩进代码和取消缩进
Alt+M	打开模块代码，先选中模块，然后按下此快捷键，会帮你打开改模块的 py 源码供浏览
Alt+C	打开类浏览器，方便在源码文件中的各个方法体之间切换
F5	运行程序

1.2.4 第一个 python 源程序

```
1. print('Hello,world!')
```

将源代码保存到：E:\python\first-programme.py

在 IDLE 中单击 F5 或者 run-->run module 执行这个源程序

第一个 Python 程序中需要注意的小要点：

1. 不要在程序中，行开头处增加空格。空格在 Python 中有缩进的含义。
 2. 符号都是英文符号，不是中文。比如：(, ”
- 下面我们全程讲解第一个程序编码过程。

1. Windows 命令行编写代码 (Console)

通过上节基础性的学习，各位可能感觉在命令行 (Console) 控制台中编写代码觉得有点枯燥，感觉效率不是很高。不过这是为了照顾零基础初学者，让大家都打下个扎实的基础，后面我们会慢慢过渡到使用 IDE (Integrated Development Environment) 集成开发环境来高效率地编写 Python 代码。这里来看一个简单的例子，读者参考上节的步骤，进入 Console 界面，开始敲入以下代码。

```
1. #直接给变量赋值，无需先定义变量类型
2. var01="First console python code"
3. var02="Python execute successfully"
4.
5. #打印输出函数
6. print(var01)
7. print(var02)
```

Console 的两种执行方式如下：

第一种方式：进入 Console 界面后，一边编写代码，一边执行。

第二种方式：将 Python 代码以文本编辑器编写好并保存，然后在 Console 当中进入 Python 代码所在文件夹，只需要输入 Python 文件名，即可解释执行 Python 代码。命令格式为：python filename.py，其中 filename 是 Python 文件的名字。

实际上，在 Linux 系统中，如果一字不差的执行书中代码，不会顺利出来上述结果的，可能会提示出如下错误信息：

```
C:\>python 0101.py
File "0101.py", line 3
SyntaxError: Non-ASCII character '\xb1' in file 0101.py on line 3, but no encoding
declared; see http://python.org/dev/peps/pep-0263/ for details
```

在未来的程序学习之旅，调试程序也是需要学习和锻炼的一项基本功。看懂上述错误的原因了吗？这里是因为程序代码中有中文注释。但是，Python 中默认采用 ASCII 编码格式，无法处理汉字。因此，需要添加一行支持中文的代码说明：`#-*- coding:utf-8 -*-`。改进后的程序代码如下面程序所示，这时程序就不会再出现错误提示了。

```
1. #-*- coding:utf-8 -*-
2. #直接给变量赋值，无需先定义变量类型
3. var01="First console python code"
4. var02="Python execute successfully"
5.
6. #打印输出函数
7. print(var01)
8. print(var02)
```

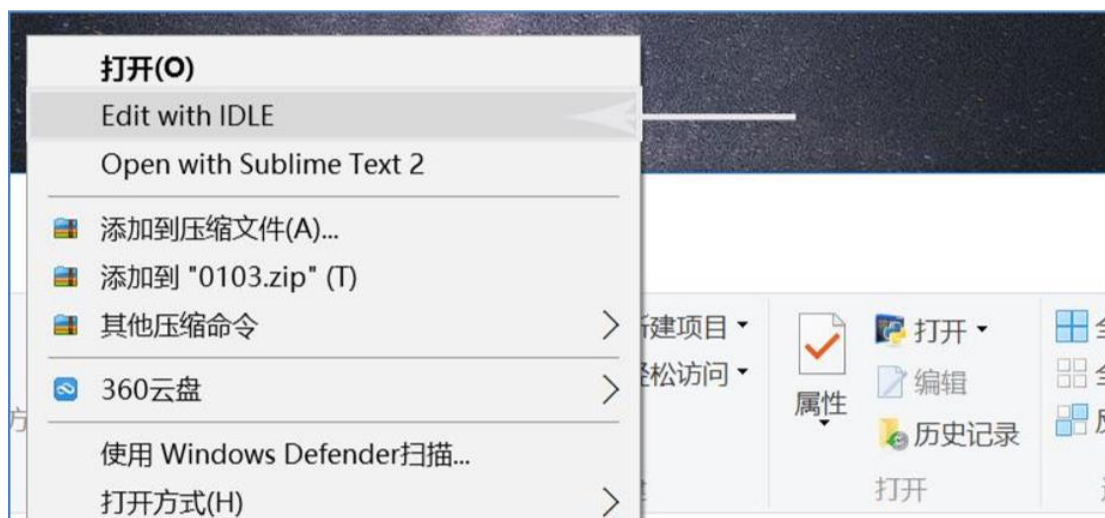
2. IDLE 编写代码

我们讲了上述 Python 代码执行方式后，我们现在介绍一下 Python 官方自带的原生编辑器。这款原生编辑器比起 Console 界面编程方式在用户体验方面舒适方便了一些。Python 自带的一款集成开发环境：IDLE。它可以让初学者方便的掌握基本的 Python 代码编写与调试工作。我们现在开始了解如何操作这款工具吧。



读者在使用 Python 的 IDLE 时会发现，每次编辑完代码完成后，按回车键后并不会马上执行程序并输出结果，而是第二次按回车键时候才执行了程序结果。这是因为编辑器认为可能还会输入下一条代码，除非连续按二次回车，系统才会知道没有代码编辑了，需要解释执行代码（当然，如果输入 print 语句，会马上执行）。通常，我们会在一个文本编辑器中编写 Python 代码，Python 的 IDLE 自带有编辑器，我们选中要编码的 Python 文件，鼠标右键打开：Edit with IDLE，然后就可以在界面里敲入代码了。代码编写完成之后，

单击【菜单】|【Run】|【Run Module】(或 F5)，就可在新弹出的 Python 3.8 shell 窗口查看程序运行结果了。



1.3 程序基本格式

1. 恰当的空格，缩进问题
 - (1) 逻辑行首的空白（空格和制表符）用来决定逻辑行的缩进层次，从而用来决定语句的分组。
 - (2) 语句从新行的第一列开始
 - (3) 缩进风格的统一：
 - a. 每个缩进层次使用 单个制表符 或四个空格（IDE 会自动将制表符设置成 4 个空格）
 - b. Python 用缩进而不是 {} 表示程序块
2. Python 区分大小写
3. 注释
 - (1) 行注释
每段注释前加#。当解释器看到#，则忽略这一行#后边的内容
 - (2) 段注释
使用连续三个单引号('')。当解释器看到'''，则会扫描到下一个'''，然后忽略他们之间的内容。

1.4 开始学习图形化程序设计

为了让初学者更加容易接受编程，我们这里先从海龟画图开始讲解。这样，大家在不接触其他编程概念时，就能开始做出一些简单的效果。提高兴趣，寓教于乐。

```
1. import turtle          #导入 turtle 模块
2. turtle.showturtle()    #显示箭头
3. turtle.write("施想")    #写字符串
4. turtle.forward(300)     #前进 300 像素
5. turtle.color("red")     #画笔颜色改为 red
6. turtle.left(90)         #箭头左转 90 度
7. turtle.goto(0,50)       #去坐标 (0,50)
8. turtle.goto(0,0)
```

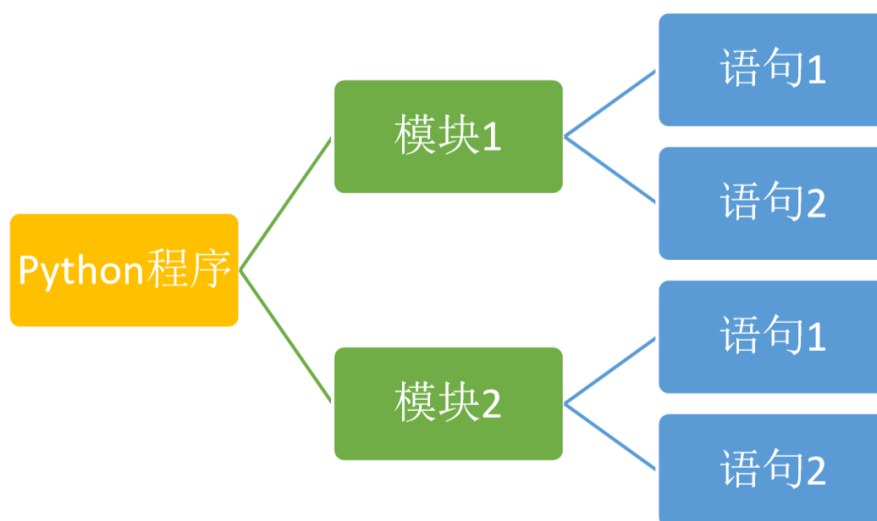
```
9. turtle.penup()          #抬笔。返样，路径就不会画出来
10. turtle.goto(0,300)
11. turtle.pendown()       #下笔。这样，路径就会画出空
12. turtle.circle(100)     #画圆
```

课后作业 1：完成奥运五环画图程序



2.1 编程基本概念

2.1.1 Python 程序的构成



1. Python程序由模块组成。一个模块对应python源文件，一般后缀名是：.py。
2. 模块由语句组成。运行Python程序时，按照模块中语句的顺序依次执行。
3. 语句是Python程序的构造单元，用于创建对象、变量赋值、调用函数、控制语句等。

2.1.2 Python 文件的创建和执行

前面使用的交互式环境，每次只能执行一条语句；为了编写多条语句实现复杂的逻辑，本章开始我们通过创建Python文件，并执行该文件。

在IDLE环境中，我们可以通过File→new创建Python文件，并可以编辑该文件内容。我们也可以通过File→save/save as保存文件。一般保存成扩展名为py的文件。

需要执行编辑好的文件，可以用快捷键 F5 或者点击 Run-->Run module。

2.1.3 使用注释#

注释是程序中会被Python解释器忽略的一段文本。程序员可以通过注释记录任意想写的内容，通常是关于代码的说明。

Python 中的注释只有单行注释，使用#开始知道行结束的部分。

1. #注释是个好习惯，方便自己方便他人
2. a = [10,20,30] #生成一个列表对象，变量 a 引用了这个变量

2.1.4 使用\行连接符

一程序长度是没有限制的，但是为了可读性更强，通常将一行比较长的程序分为多行。这是，我们可以使用\行连接符，把它放在行结束的地方。Python 解释器仍然将它们解释为同一行。

```
test.py ===== RESTART: C:/Users/27176/Desktop/test.py (3.8.1)
[10, 20, 30, 40, 50, 60, 70, 80]
>>>
```

test.py - C:/Users/27176/Desktop/test.py (3.8.1)
File Edit Format Run Options Window Help

```
a=[10, 20, 30, \
40, 50, 60, \
70, 80]
print(a)
```

列表

```
===== RESTART: C:/Users/27176/Desktop/test.py (3.8.1)
abcdefghijklmnopqrstuvwxy
>>>
```

test.py - C:/Users/27176/Desktop/test.py (3.8.1)
File Edit Format Run Options Window Help

```
a='abcde\
fghij\
klmnop\
qrstuv\
wxyz'
print(a)
```

字符串

2.1.5 对象

Python 中，一切皆对象。每个对象由：标识 (identity)、类型 (type)、value (值) 组成。

1. 标识用于唯一标识对象，通常对应于对象在计算机内存中的地址。使用内置函数 id(obj)

可返回对象 obj 的标识。

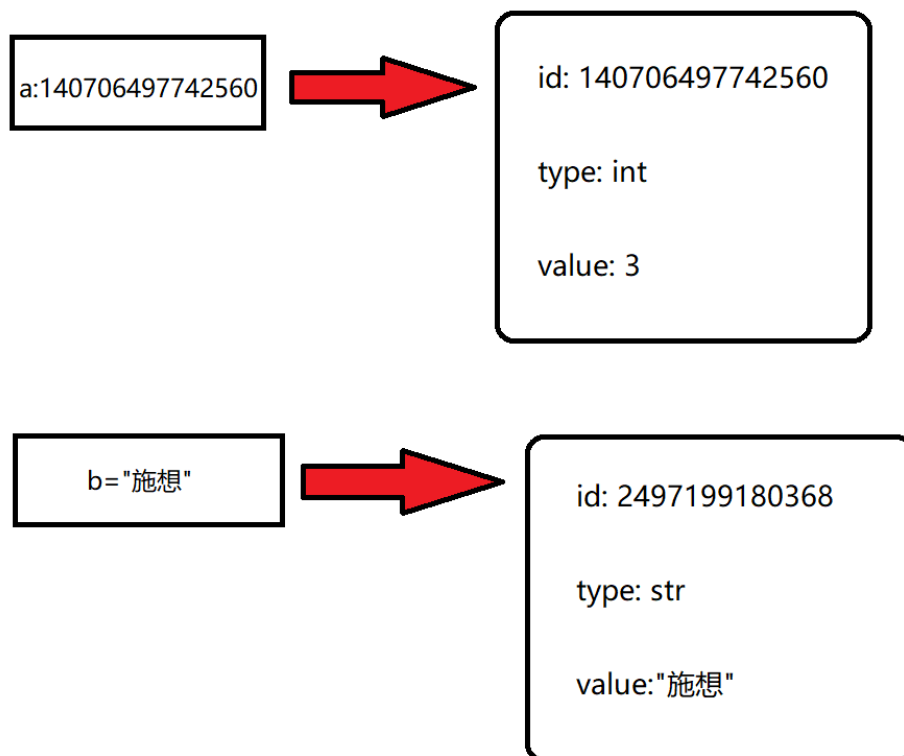
2. 类型用于表示对象存储的“数据”的类型。类型可以限制对象的取值范围以及可执行的操作。可以使用 `type(obj)` 获得对象的所属类型。

3. 值表示对象所存储的数据的信息。使用 `print(obj)` 可以直接打印出值。

对象的本质就是：一个内存块，拥有特定的值，支持特定类型的相关操作。

```
>>> a=3
>>> a
3
>>> id(3)
140706497742560
>>> type(3)
<class 'int'>
>>> b="施想"
>>> id(a)
140706497742560
>>> type(a)
<class 'int'>
>>> id(b)
2497199180368
>>> type(b)
<class 'str'>
```

示意图：



2.1.6 引用

在Python中，变量也成为：对象的引用。因为，变量存储的就是对象的地址。

变量通过地址引用了“对象”。

变量位于：栈内存（压栈出栈等细节，后续再介绍）。

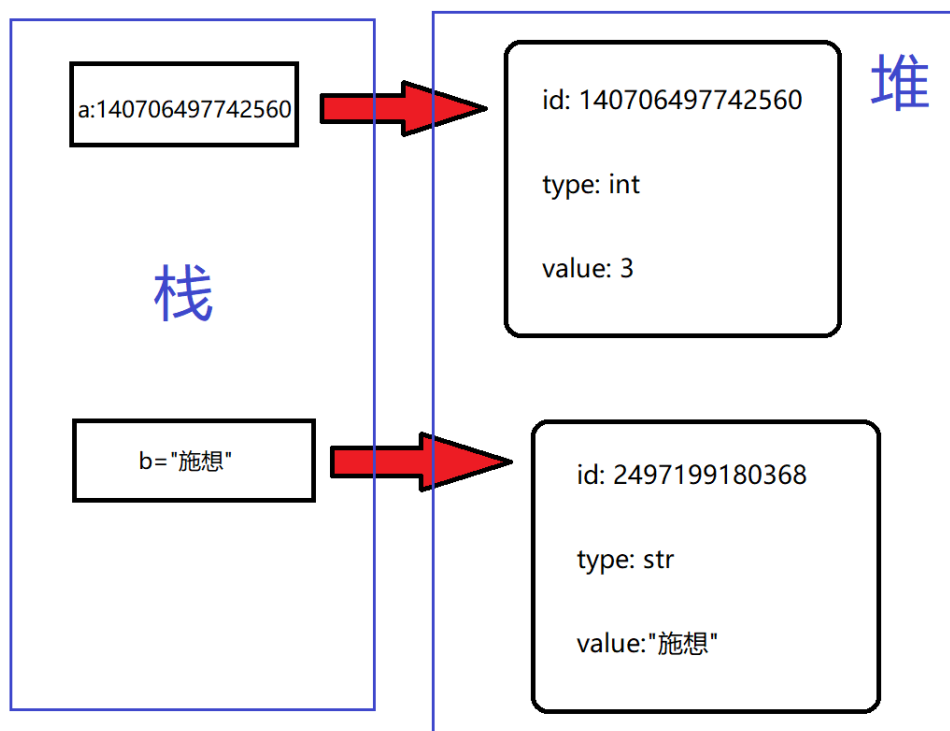
对象位于：堆内存。

- Python是动态类型语言

变量不需要显式声明类型。根据变量引用的对象，Python解释器自动确定数据类型。

- Python是强类型的语言

每个对象都有数据类型，只支持该类型支持的操作。



2.1.7 标识符

Python语法中，标识符由数字、字母、下画线组成，所有标识符不能以数字开头，且

Python标识符区分字母的大小写。以下划线开始的标识符有特殊意义，归类如下：

单下画线开头：不能直接访问类属性，需要通过接口进行访问，不能采用from xxx import *的方式。

双下画线开头：类的私有成员。

xxx 双下画线开头和结尾：特殊方法专用的标识。如__init__()代表构造函数。

标识符：用于变量、函数、类、模块等的名称。标识符有如下特定的规则：

1. 区分大小写。如：sx和SX是不同的
2. 第一个字符必须是字母、下划线。其后的字符是：字母、数字、下划线
3. 不能使用关键字。比如：if、or、while等。
4. 以双下划线开头和结尾的名称通常有特殊含义，尽量避免这种写法。比如：__init__是类的构造函数。

*使用help（）查看关键字

关键字无需可以去背，后面都会学习

Python保留字不能作为常数或变量，以及其他任何标识符名称，且都是小写字母。

Python保留字：

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

2. 2变量和简单赋值语句

2. 2. 1 变量的声明和赋值

变量的声明和赋值用于将一个变量绑定到一个对象上，格式如下：

变量名 = 表达式

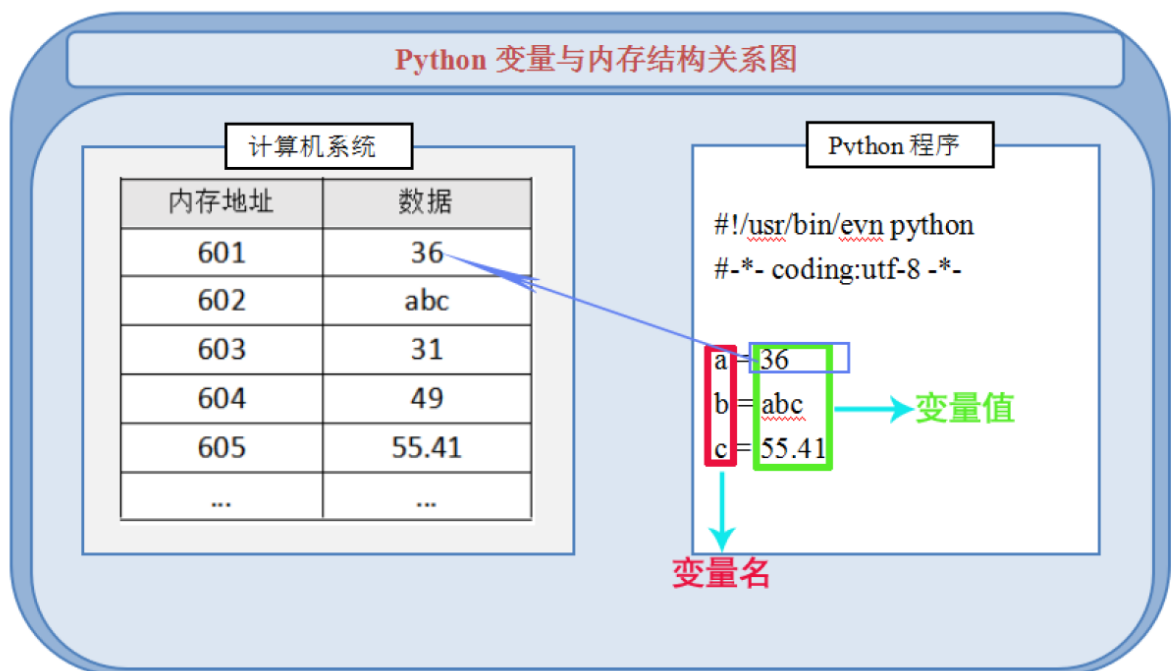
最简单的表达式就是字面量。比如：a = 123 。运行过程中，解释器先运行右边的表达式，生成一个代表表达式运算结果的对象；然后，将这个对象地址赋值给左边的变量。

【操作】变量在使用前必须先被初始化（先被赋值）

```
>>> sx
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    sx
NameError: name 'sx' is not defined
\\
```

变量sx在被使用前未做赋值，因此报错：'sx' is not defined。

变量存储在内存当中，当为变量赋值时，系统会在内存中开辟一个存储空间。变量中存放的数据都具有特定的数据类型，变量可以存储整数、小数、复数、字符等。



下面是Python变量常见类型：

1. 变量赋值
2. 数值型
3. 字符串
4. 列表
5. 元组
6. 字典
7. 集合

1. Python变量赋值

Python变量赋值不需要类型声明，直接赋值即可，且Python支持多个变量赋值，多变量赋值形式，比如：a = b = c = 12。

2. Python数值类型

Python数值类型（Number）是存储数值，Python支持四种不同的数值类型：

int（有符号整型）

float（浮点型）

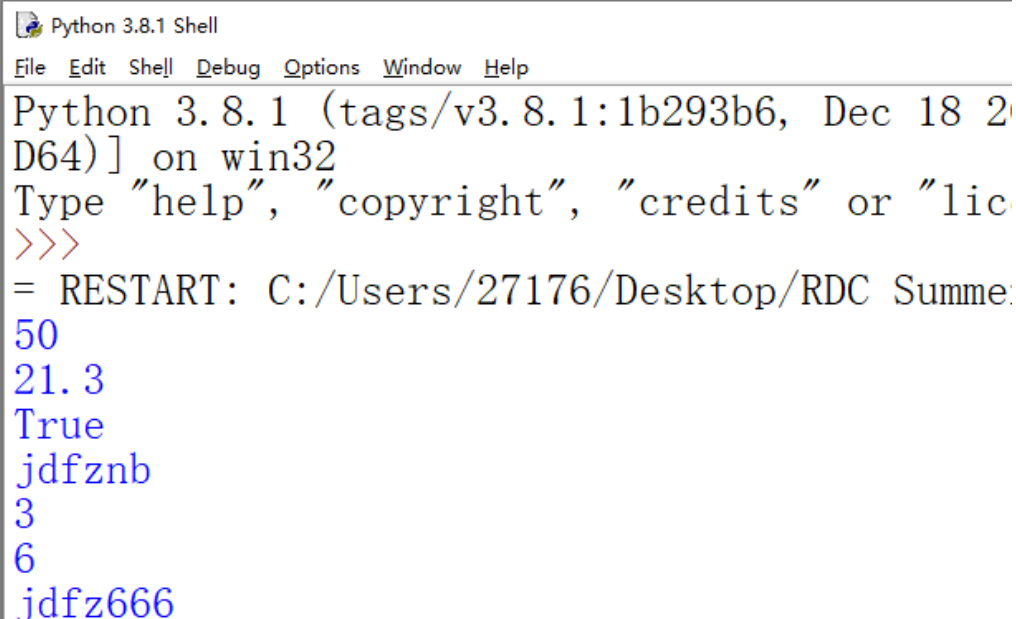
complex（复数）

bool（布尔数）

数值类型实例：

```
a = 50 #赋值整型变量
b = 21.3 #浮点型变量
c = True #布尔型变量
d = "jdfznb" #字符串型变量
e, f, g = 3, 6, "jdfz666" #多个变量赋值

print(a);print(b);print(c);print(d);print(e)
```



```
Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 10:23:02) [AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more
>>>
= RESTART: C:/Users/27176/Desktop/RDC Summe
50
21.3
True
jdfznb
3
6
jdfz666
```

int	float	complex
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3e+18	.876j
-0490	-90.	-.6545+0J
-0x260	-32.54e100	3e+26J
0x69	70.2E-12	4.53e-7j

字符串类型（String）是由数字、字母、下划线组成的一串字符。字符串类型用来存储各类文本数据。Python的字符串取子串的话，有两种取值方式：

- 从左到右，以0下标索引开始
- 从右到左，以-1下标索引开始

```

=====
ion 2020/str_example.py =====
jdfz666
j
6
jdfz666
z66
66
666
dfz666
jdfz666jdfz666
jdfz666 jdfz666
I love jdfz, jdfz666
>>>
File Edit Format Run Options Window Help
a = "jdfz666"
print(a)
print(a[0])
print(a[-1])
print(a[:])
print(a[3:6])
print(a[-3:-1])
print(a[-3:])
print(a[1:])
print(a * 2)
print((a+" ") * 2)
print("I love jdfz," + a)

```

温馨提示:Python 双引号与单引号的意义相同，都可表示为字符串。

字符串双引号之前加上字母u，Python会将其解释为Unicode字符串。

字符串格式化是将字符串经过处理后显示出希望的目标形式。

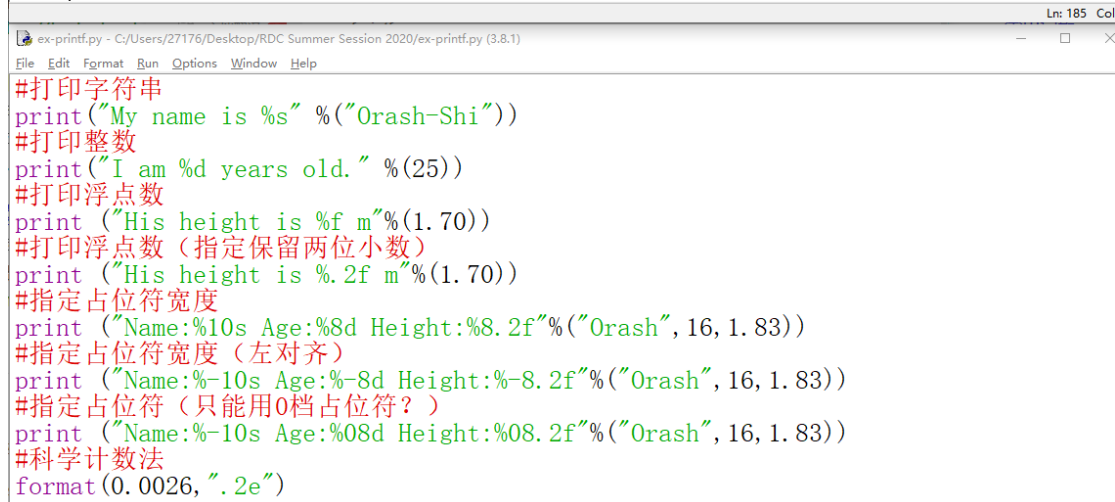
其表达式语法：format_string % string_to_convert..... %左边为格式化要求，%为要格式化的内容。格式化符号相关说明如表：

Python中转义字符与其它高级程序设计语言区别不大，主要用来格式化不显示的字符，例如缩进，换行，回车等等，转义字符如下表所示：

格式	描述
%%	百分号标记
%c	字符及其ASCII码
%s	字符串
%d	有符号整数(十进制)
%u	无符号整数(十进制)
%o	无符号整数(八进制)
%x	无符号整数(十六进制)
%X	无符号整数(十六进制大写字符)
%e	浮点数字(科学计数法)
%E	浮点数字(科学计数法, 用E代替e)
%f	浮点数字(用小数点符号)
%g	浮点数字(根据值的大小采用%e或%f)
%G	浮点数字(类似于%g)
%p	指针(用十六进制打印值的内存地址)
%n	存储输出字符的数量放进参数列表的下一个变量中

示例：

```
===== RESTART: C:/Users/27176/Desktop/RDC Summer Session 2020/ex-printf.py =====
My name is Orash-Shi
I am 25 years old.
His height is 1.700000 m
His height is 1.70 m
Name:      Orash Age:      16 Height:      1.83
Name:Orash Age:16      Height:1.83
Name:Orash Age:00000016 Height:00001.83
>>> |
```



```
ex-printf.py - C:/Users/27176/Desktop/RDC Summer Session 2020/ex-printf.py (3.8.1)
File Edit Format Run Options Window Help
#打印字符串
print("My name is %s" %("Orash-Shi"))
#打印整数
print("I am %d years old." %(25))
#打印浮点数
print("His height is %f m"%(1.70))
#打印浮点数（指定保留两位小数）
print("His height is %.2f m"%(1.70))
#指定占位符宽度
print("Name:%10s Age:%8d Height:%8.2f"%("Orash", 16, 1.83))
#指定占位符宽度（左对齐）
print("Name:%-10s Age:%-8d Height:%-8.2f"%("Orash", 16, 1.83))
#指定占位符（只能用0档占位符？）
print("Name:%-10s Age:%08d Height:%08.2f"%("Orash", 16, 1.83))
#科学计数法
format(0.0026, ".2e")
```

4. Python列表（List）

列表可以表示Python大多数的集合类数据结构，采用[]标识，是Python最常用的符合数据类型。列表支持字符、数字、字符串、列表（嵌套）等。列表实例代码：

```
===== RESTART: C:/Users/27176/Desktop/RDC Summer Session 2020/list_name.py =====
['y', 'y', 'q', ' ', ' ', 6, 'abc', 88.9]
y
['q', ' ', ' ', 6]
['y', 'y', 'q', ' ', ' ', 'connecting', 'me', 'please', '?']

list_name.py - C:/Users/27176/Desktop/RDC Summer Session 2020/list_name.py (3.8.1)
File Edit Format Run Options Window Help
list1=['y', 'y', 'q', ' ', ' ', 6, 'abc', 88.9]
list2=['connecting', 'me', 'please', '?']

print(list1)
print(list1[0])
print(list1[2:5])
print(list1[0:4] + list2[0:4])
```

5. Python元组（Tuple）

元组用（）标识，内部元素用逗号分隔。元组内的元素不可以修改，意味着不可以二次赋值。示例：

```
>>> tup1=(520)
>>> type(tup1)
<class 'int'>
>>> tup1=(520,)
>>> type(tup1)
<class 'tuple'>
```

```

>>> tup_a=(1, 2, 3, 'a', 'b', 'c', 'abc')
>>> tup_b=('connecting', 'me')
>>> print(tup_a)
(1, 2, 3, 'a', 'b', 'c', 'abc')
>>> print(tup_b)
('connecting', 'me')
>>>
>>> print(tup_a[0])
1
>>> print(tup_a[1:3])
(2, 3)
>>> print(tup_a + tup_b)
(1, 2, 3, 'a', 'b', 'c', 'abc', 'connecting', 'me')
>>>

```



首先，通过对Python基础的学习，分别对List和Tuple两类数据类型做知识点梳理，我们最终采用二分法的形式对二者的区别与联系加以分析。我们这里采纳了思维导图的方式，更加容易理解，人们通常对图形的识别和印象是比较深刻，过程简单吧。

这里，将问题又划分为两大类：一类是偏向宏观方面，比如数据类型是否属于Sequence序列簇，是否是基本数据类型；另一类是更加具体的，比如数据的函数性质，操作符操作权限以及系统内存分配等。

相信同学们通过上述图示的分析，对于Python当中其它数据类型的掌握和运用会更加轻车熟路。

6. Python字典（Dictionary）

字典数据类型是有序的对象集合。字典的存取采用键-值存取的方式，即通过键取值。字典用{}标识。

关于键-值对存储形式在计算机科学当中应用非常广泛。程序访问键-值对的‘键’，系统会根据‘键’来获取对应的‘值’，‘键’与‘值’是一一对应关系，且‘键’不能重复。当我们的Python程序访问字典的每一个键时，根据其去查找唯一对应的值。

其工作原理如下图所示：

key	value
China	100
America	90
Australia	abc
England	33.21
France	dfg
Russia	- 33.2
Germany	90

key值不能相同

value可以相同

字典（Dictionary）相关代码实例如下图所示。

```

===== RESTART: C:/Users/27176/AppData/Local/Programs
dict['Name']: Orash
dict['Age']: 16
dictionary_ex.py - C:/Users/27176/AppData/Local/Programs/Python/Python38/dictionary_ex.py (3.8.1)
File Edit Format Run Options Window Help
dict = {'Name': 'Orash', 'Age': 16, 'Class': 'Six'}

print ("dict['Name']: ", dict['Name'])
print ("dict['Age']: ", dict['Age'])

```

7. Python集合（Set）

集合（Set）是用于表示相互之间无序的一组元素。集合在算术上的运算包括交集、并集、补集等。同时，Python的集合又分为二类：

- 普通集合：实现交集、并集、补集操作，通过关键字set实现。
- 不可变集合：初始化后不可以改变，通过关键字frozenset实现

集合（Set）相关操作符如下图所示：

表 1-3 Set 集合操作表

Set 类型	
in	判断包含关系
notin	判断不包含关系
= =	判断等于
!=	判断不等于
<	判断绝对子集关系
<=	判断非绝对子集关系
>	判断绝对超集关系
>=	判断非绝对超集关系
&	交运算
	并运算
-	差运算
^	对称差运算
=	并运算并赋值
&=	交运算并赋值
- =	差运算并赋值
^ =	对称运算并赋值

Set类型相关代码实例如下：

```
1 { 'a', 'c', 'b' }
>>>

set_ex.py - C:/Users/27176/AppData/Local/Programs/Python/Python36-32/Scripts/python.exe
File Edit Format Run Options Window Help
a= { 'a', 'b', 'c' }
print(a)
```

8. Python数据类型转换

数据类型的转换是编程当中经常会使用到的方法，我们只需要将数据类型作为函数名便可。关于一共有多少个转换函数可查阅相关问答，无需强硬的记忆。我们需要学会如何查找，如何看懂使用说明，这是需要加强的学习方法。

语法格式：函数名（对象）

例如：float(36) 含义：将整型数字36转化成浮点型

2.2.2 删除变量和垃圾回收机制

可以通过del语句删除不在使用的变量。

【操作】 删除变量示例

```
>>> a=123
>>> del a
>>> a
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    a
NameError: name 'a' is not defined
```

如果对象没有变量引用，就会被垃圾回收器回收，清空内存空间。

2.2.3 链式赋值

链式赋值用于同一个对象赋值给多个变量。

x=y=123 相当于：x=123；y=123

2.2.4 系列解包赋值

系列数据赋值给对应相同个数的变量（个数必须保持一致）

```
>>> a,b,c=4,5,6  相当于：a=4;b=5;c=6
```

【操作】使用系列解包赋值实现变量交换

```
>>> a, b=1, 2
>>> a, b=b, a
>>> print(a, b)
2 1
>>>
```

2.2.5 常量

Python不支持常量，即没有语法规则限制改变一个常量的值。我们只能约定常量的命名规则，以及在程序的逻辑上不对常量的值作出修改。

```
>>> MAX_SPEED = 120
```

```
>>> print(MAX_SPEED)
```

```
120
```

```
>>> MAX_SPEED = 140    #实际是可以改的。只能逻辑上不做修改。
```

```
>>> print(MAX_SPEED)
```

```
140
```

2.3 最基本内置数据类型和运算符

2.3.1 数据类型

物以类聚，人以群分，数据亦是如此。数据类型是一个集合以及定义在这个集合上的一组操作。数值类型必然都是数字，字符串类型必然都是字符串。当然，还会有一些高级的数据类型，初学者可能不是很容易理解，但相信通过我们的讲解会顺利掌握。

1. 类型的定义

Python当中不同的数据类型，其数据取值范围明确了数据的精度表示范围。当涉及到数据类型定义时，我们主要会关注如下几点：

- 引进数据类型的作用
- 是什么数据类型呢
- 数据取值范围的决定因素

（1）引进数据类型的作用

计算机的0与1代码最终存放的是整数、浮点数、字符串还是布尔值，因此要引入数据类型的概念，定义为整数类型，那么其存储的值就是整数。不然，计算机就不知道这个数据到底是什么数据类型，不会处理了。例如，存储单元内的100的代表整数100，还是ASCII码的字符'd'？当然，数据类型也是可以人为转换的嘛，可以把整数类型转化为浮点数、字

字符串等类型，因为毕竟任何类型的数据最终都是要转化为计算机底层的0与1代码。

（2）是什么数据类型呢

Python数据类型分为基本数据类型、高级数据类型二大类。任何数据最终都要归类到某一个具体的数据类型，不然计算机就无法继续工作啦。因此，我们要养成一个思维习惯，每当编程时，首先要明确的就是各种数据的类型，至少心中要有数，对数据有一个明晰的归类。无论是普通数据还是大数据，最终也是转化为计算机的0与1代码，交给计算机来处理。

（3）数据的取值范围由哪些因素决定

数据的取值范围？首先，要知道数据最终是要存储到计算机硬件上的。计算机硬件又分数据的取值范围？首先，要知道数据最终是要存储到计算机硬件上的。计算机硬件又分为32位与64位机器。那么就简单了，整型在32位机器上的取值范围是 -2^{32} -- $2^{32}-1$ ，64位机器上的取值范围是 -2^{64} -- $2^{64}-1$ 。值得一提的是，python语言中的整型没有限制取值范围，理论上是可以任意大数值的。但是，同时我们要注意计算机内存是有限度的，不可能任由我们取无限大的数值。

2. 运算符

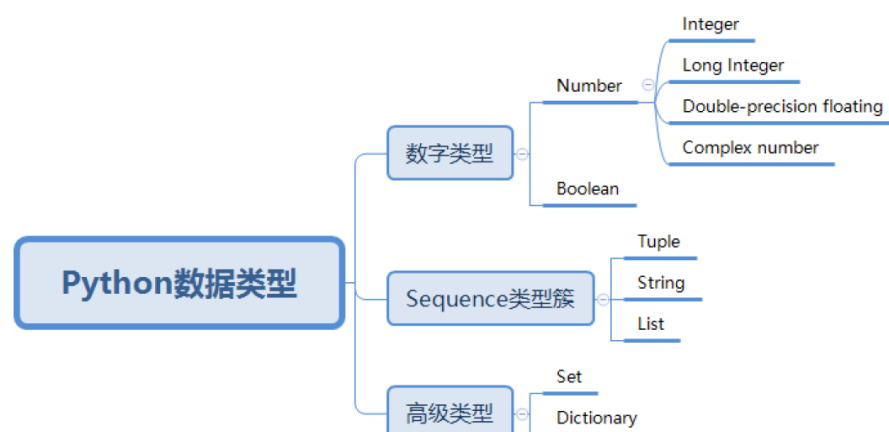
Python和其它程序设计语言一样，拥有丰富的运算符，主要包括：

- 位运算符
- 算术运算符
- 比较运算符
- 逻辑运算符
- 赋值运算符
- 成员运算符
- 身份运算符
- 运算符优先级

3. 类型操作符

以类型操作作为基准，我们将Python数据类型总体上分为三大类，那么，类型操作符也主加以区别并归类：

- 数字类型操作符
- Sequence类型簇操作符
- 高级数据类型操作符



2.3.2 数字和基本运算符

Python支持整数(如: 50,520)和浮点数(如: 3.14,10.0, 1.23e2), 我们可以对数字做如下运算。

运算符	描述	实例
+	加 - 两个对象相加	a + b 输出结果 31
-	减 - 得到负数或是一个数减去另一个数	a - b 输出结果 -11
*	乘 - 两个数相乘或是返回一个被重复若干次的字符串	a * b 输出结果 210
/	除 - x 除以 y	b / a 输出结果 2.1
%	取模 - 返回除法的余数	b % a 输出结果 1
**	幂 - 返回x的y次 ^{关闭}	a**b 为10的21次方
//	取整除 - 向下取接近商的整数	<pre>>>> 9//2 4 >>> -9//2 -5</pre>

```
>>> divmod(10, 5)
(2, 0)
>>> divmod(10, 3)
(3, 1)
>>> |
```

divmod() 是一个函数, 我们以后会详细介绍。他返回的是一个元组(后续将会学习)。

2.3.3 整数

Python中, 除10进制, 还有其他三种进制:

·0b或0B, 二进制0 1

·0o或0O, 八进制0 1 2 3 4 5 6 7

·0x或0X, 十六进制0 1 2 3 4 5 6 7 8 9 a b c d e f

这三种进制可以非常方便的进行“位运算”操作。位运算知识后续将会介绍

【操作】测试不同进制

```
>>> 12
12
>>> 0b101
5
>>> 0o10
8
>>> 0xff
255
>>> 0xf
15
>>> 0x10
16
```

使用int()实现类型转换:

1. 浮点数直接舍去小数部分。如：`int(9.9)`结果是：9
2. 布尔值True转为1，False转为0。如：`int(True)`结果是1
3. 字符串符合整数格式（浮点数格式不行）则直接转成对应整数，否则报错。

自动转型:

整数和浮点数混合运算时，表达式结果自动转型成浮点数。比如：2+8.0的结果是10.0

整数可以有多大？

Python2中, int是32位, 可以存储从-2147483648到2147483647的整数 (约 ± 21 亿)。

Long类型是64位，可以存储： -2^{63} -- $2^{63}-1$ 之间的数值。

Python3中，int可以存储任意大小的整数，long被取消。我们甚至可以存储下面的值：

[illegible]

Googol也是Google最初的名字，这也是Google最初的含义。

Python3中可以做超大数的计算，而不会造成“整数溢出”，这也是Python特别适合科学运

算的特点。

2.3.4 浮点数

浮点数，称为float。

浮点数用形式的科学计数法表示。比如：3.14，表示成：314E-2或者314e-2。

这些数字在内存中也是按照科学计数法存储。

2.3.5 类型转换和四舍五入

1. 类似于int()，我们也可以使用float()将其他类型转化成浮点数。
2. 整数和浮点数混合运算时，表达式结果自动转型成浮点数。比如：2+8.0的结果是10.0
3. round(value)可以返回四舍五入的值

注：但不会改变原有值，而是产生新的值

2.3.6 增强型赋值运算符

运算符+、-、*、/、//、**和%和赋值符=结合可以构成“增强型赋值运算符”。

`a = a + 1` 等价于：`a += 1`

运算符	例子	等价
<code>+=</code>	<code>a += 2</code>	<code>a = a + 2</code>
<code>-=</code>	<code>a -= 2</code>	<code>a = a - 2</code>
<code>*=</code>	<code>a *= 2</code>	<code>a = a * 2</code>
<code>/=</code>	<code>a /= 2</code>	<code>a = a / 2</code>
<code>//=</code>	<code>a //= 2</code>	<code>a = a // 2</code>
<code>**=</code>	<code>a **= 2</code>	<code>a = a ** 2</code>
<code>%=</code>	<code>a %= 2</code>	<code>a = a % 2</code>

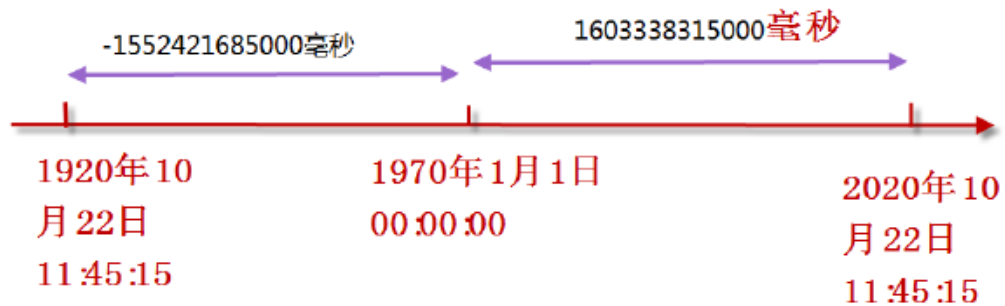
注意：“+=”中间不能加空格！

2.3.7 时间的表示

计算机中时间的表示是从“1970年1月1日00:00:00”开始，以毫秒（1/1000秒）进行计

算。我们也把1970年这个时刻成为“unix时间点”。

这样，我们就把时间全部用数字来表示了。



python中可以通过`time.time()` 获得当前时刻，返回的值是以秒为单位，带微秒（1/1000毫秒）精度的浮点值。

2.3.8 布尔值

Python2中没有布尔值，直接用数字0表示False, 用数字1表示True。

```
>>> a = True
>>> b = 3
>>> a+b
4
```

2.3.9 比较运算符

所有比较运算符返回1表示真，返回0表示假。这分别与特殊的变量True和False等价。

以下假设变量a为15，变量b为30：

==	等于 - 比较对象的值是否相等	(a == b) 返回 False。
!=	不等于 - 比较两个对象的值是否不相等	(a != b) 返回 true。
>	大于 - 返回 x 是否大于 y	(a > b) 返回 False。
<	小于 - 返回 x 是否小于 y。	(a < b) 返回 true。
>=	大于等于 - 返回 x 是否大于等于 y。	(a >= b) 返回 False。
<=	小于等于 - 返回 x 是否小于等于 y。	(a <= b) 返回 true。

2.3.10 逻辑运算符

运算符	格式	说明
or 逻辑或	x or y	x 为 true，则不计算 y，直接返回 true x 为 false，则返回 y
and 逻辑与	x and y	x 为 true，则返回 y 的值 x 为 false，则不计算 y，直接返回 false
not 逻辑非	not x	x 为 true，返回 false x 为 false，返回 true

同一运算符用于比较两个对象的存储单元，实际比较的是对象的地址。

运算符	描述
is	is 是判断两个标识符是不是引用同一个对象
is not	is not 是判断两个标识符是不是引用不同对象

is 与 == 的区别：

is 用于判断两个变量引用对象是否为同一个，既比较对象的地址。

== 用于判断引用变量引用对象的值是否相等，默认调用对象的 `__eq__()` 方法。

2.3.11 整数缓存问题

Python 仅仅对比较小的整数对象进行缓存（范围为 `[-5, 256]`）缓存起来，而并非是所有整数对象。需要注意的是，这仅仅是在命令行中执行，而在 Pycharm 或者保存为文件执行，结果是不一样的。

的，这是因为解释器做了一部分优化(范围是[-5,任意正整数])。

•总结

- 1、is 比较两个对象的 id 值是否相等，是否指向同一个内存地址；
- 2、== 比较的是两个对象的内容是否相等，值是否相等；
- 3、小整数对象[-5,256]在全局解释器范围内被放入缓存供重复使用；
- 4、is 运算符比 == 效率高，在变量和None进行比较时，应该使用 is。

2.3.12 基本运算符

我们在前面讲解了“+”、“-”、“*”、“/”、“//”、“%”等运算符，这里我们继续讲解一些其他运算符，并进行学习和测试。

1. 比较运算符可以连用，并且含义和我们日常使用完全一致。

```
>>> a = 4
>>> 3 < a < 10
True
```

2. 位操作

```
>>> a=0b11001
>>> b=0b01000
>>> c=a|b
>>> bin(c)
'0b11001'
>>> bin(c&b)
'0b1000'
>>> bin(c^b)
'0b10001'
>>> a=3
>>> bin(a)
'0b11'
>>> a<<2
12
>>> a=8
>>> a>>1
4
```

3. 加法操作

- (1) 数字相加 3+2 ==> 5
- (2) 字符串拼接“3”+“2”==> “32”
- (3) 列表、元组等合并[10,20,30]+[5,10,100] ==> [10,20,30,5,10,100]

4. 乘法操作

- (1) 数字相乘3*2 ==>6
- (2) 字符串复制“sx”*3 ==>”sxsxsx”
- (3) 列表、元组等复制[10,20,30]*3 ==> [10,20,30,10,20,30,10,20,30]

2.3.13复合赋值运算符

复合赋值可以让程序更加精炼，提高效率。

运算符	描述	示例	等价于
+=	加法赋值	sum += n	sum = sum + n
	字符串拼接	a += "sxt"	a = a + "sxt"
-=	减法赋值	num1 -= n	num = num - n
*=	乘法赋值	a *= b	a = a * b
/=	浮点除赋值	a/=b	a = a / b
//=	整数除赋值	a//=b	a = a//b
%=	取余赋值	a%=b	a = a % b
=	幂运算赋值	a=2	a = a**2
<<=	左移赋值	a<<=2	a = a<<2
>>=	右移赋值	a>>=2	a = a>>2
&=	按位与赋值	a&=b	a = a&b
=	按位或赋值	a =b	a=a b
^=	按位异或赋值	a^=b	a = a^b

注：与C和JAVA不一样，Python不支持自增(++)和自减(--)

2.3.14运算符优先级问题

如下优先级，从高到低。

运算符	描述
**	指数 (最高优先级)
~	按位翻转
* / % //	乘, 除, 取模和取整除
+ -	加法减法
>> <<	右移, 左移运算符
&	位 'AND'
^	位运算符
<= < > >=	比较运算符
<> == !=	等于运算符
= %= /= //= -= += *= **=	赋值运算符
is is not	身份运算符
in not in	成员运算符
not or and	逻辑运算符

实际使用中，记住如下简单的规则即可，复杂的表达式一定要使用小括号组织。

1. 乘除优先加减
2. 位运算和算术运算>比较运算符>赋值运算符>逻辑运算符

2.3.15数据类型转换

与C++、Java等高级程序设计语言一样，Python语言同样也支持数据类型转换。我们这里列举了常见的Python类型转换。

int(x[,base])	将 x 转换为一个整数
long(x [,base])	将 x 转换为一个长整数
float(x)	将 x 转换到一个浮点数
complex(real [,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效 Python 表达式,并返回一个对象
Complex(A)	将参数转换为复数型
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
set(s)	转换为可变集合
dict(d)	创建一个字典。d 必须是一个序列 (key,value)元组
frozenset(s)	转换为不可变集合
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为 Unicode 字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串