Python语言基础与应用-01

陈斌 gischen@pku.edu.cn

目录

课程主讲教师

陈斌 博士

副教授，北京大学遥感与地理信息系统研究所

本科生课程：《数据结构与算法Python版》、《离散数学》、《地球与人类文明》

研究生课程：《空间数据库》、《开源空间信息软件》

研究方向

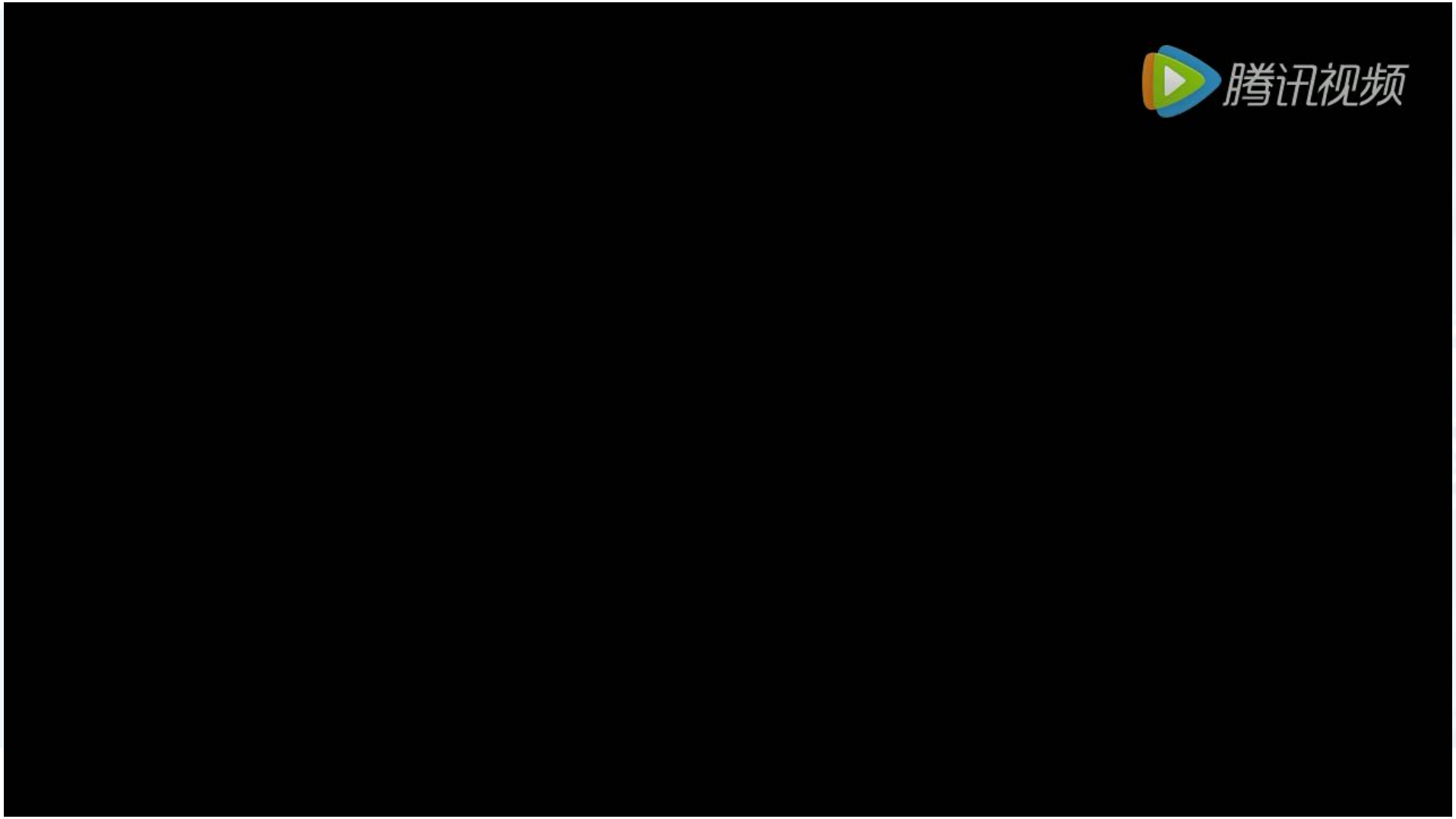
虚拟地理环境

空间信息分布式计算

联系方式

- **Email**
  gischen@pku.edu.cn
- **QQ / 微信**
  2205050
- **课程网站：**
  gis4g.pku.edu.cn/course/pythonlang
- **微信公众号**
  chbpku

What Most School Don't Teach



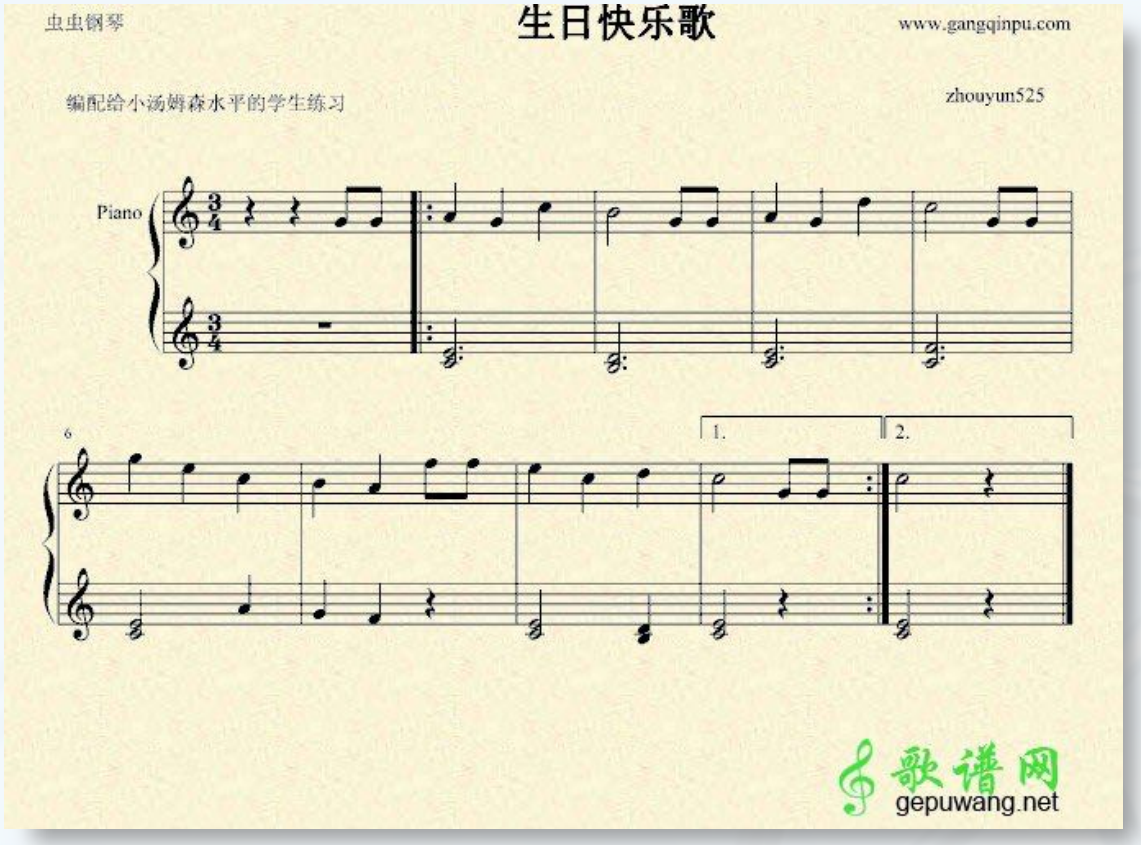
程序是什么？ #程序是菜谱

- 预热烤箱至175度
- 将面粉、苏打、盐、肉桂粉、姜粉、丁香粉混合过筛
- 准备大碗，加入黄油和糖粉，打发
- 打入鸡蛋、水和蜂蜜，搅拌
- 加入过筛混合物
- 取核桃大小面团，卷一层糖，压扁
- 放进烤箱烤8-10分钟

```
import Oven, Sifter, Bowl

oven = Oven()
bowl = Bowl()
sifter = Sifter()
oven.preheat(175)
ingredients = sifter.sift(
    [flour, ginger, baking soda,
     cinnamon, cloves, salt])
mixture = bowl.add([margarine, sugar])
while not mixture.is_light_fluffy():
    mixture = bowl.cream()
bowl.add([egg, water, molasses])
bowl.stir()
bowl.add(ingredients)
bowl.stir()
dough = bowl.get(walnut_size)
dough.rollwith([sugar])
dough.flatten()
oven.add(dough)
oven.heat(8)
if not dough.welldone():
    oven.heat(2)
```

程序是什么？ #程序是电影脚本 #程序是乐谱

镜号	景别	拍摄角度	时长	声音	内容	备注
1	全景	平拍、右斜侧	5 秒	小孩子哭啼声	小孩躺在地上哇哇大哭	三四岁的小女孩
2	全景	仰拍、右斜侧	6 秒	孩子依旧哭啼，声音越来越大	夫妻两吵架，丈夫气急败坏摔桌子（茶几）上的东西	以孩子的角度拍摄
3	特写	俯拍、左斜侧	2 秒	孩子依旧哭啼	地上摔掉的东西	以孩子的角度拍摄
4	特写	平拍、正拍	2 秒	孩子停止哭啼	孩子惊吓的表情，停止哭啼	房屋一片安静
5	全－中－近－全－远	平拍、左斜侧－正侧－右斜侧－背面	6 秒	只有丈夫离开的脚步声	丈夫甩掉东西后生气的离开家，小孩子的视线跟随着丈夫的离开而移动	机位可设在小孩背后
6	近景	平拍，（左或右）斜侧	3 秒		妻子坐在沙发上轻轻的哭泣，默默的流泪	
7	全景	稍俯拍、右斜侧平－仰	10 秒		小孩子从坐的地方慢慢向桌子（茶几）爬去，很不熟练的在一盒清风卫生纸里面抽了一张，站起来，跟跟跄跄的走到妈妈身边，把	爬 5 秒、抽纸 3 秒、递给妈妈 3 秒 俯拍小孩爬、平拍



如何用程序解决问题？求一些数的和

非程序思维是这样

- 有2个数

```
print (2+3)
```

- 有3个数

```
print (2+3+15)
```

- 有8个数

```
print (2+3+15+17+1+33+132+76)
```

- 有1000个数……？

程序思维是这样

- 有n个数

设置一个sum用来暂存部分和

sum = 第1个数

反复做下列工作，直到所有数完成：

　　取下一个数，累加到sum

输出sum

课程总体安排

日期	Day1	Day2	Day3	Day4
上午	Python概述和体验 【上机练习】	输入输出I/O 控制流和程序结构 【上机练习】	高级特性简介 （OO、异常、迭代器和生成器） 【上机练习】	实习项目 分组讨论 报告规划 开始开发
下午	基本数据类型 【上机练习】	基本模块介绍 （时间、算术、持久化、文件、数据库、GUI、海龟） 【上机练习】	基本模块介绍 （PIL、Flask、pygame、numpy、matplotlib、raspberry-gpio） 【上机练习】	编程开发 展示和总结
备注				

Python的历史

- 1989 年 12 月 ， Guido van Rossum为了打发圣诞节假期，开发了ABC脚本语言的后继
- Python名称来自于他喜欢的一个情景剧 Monty Python's Flying Circus
- Python是一种高级动态、完全面向对象的语言，函数、模块、数字、字符串都是对象，并且完全支持继承、重载、派生、多继承，有益于增强源代码的复用性。

Python的现在

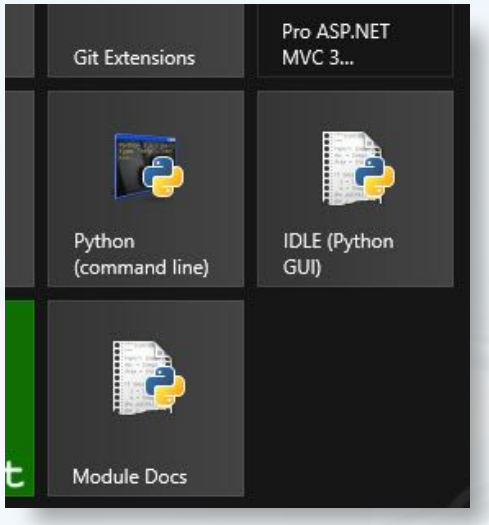
Python 编程 解释与应用

Mar 2017	Mar 2016	Change	Programming Language	Ratings	Change
1	1		Java	16.384%	-4.14%
2	2		C	7.742%	-6.86%
3	3		C++	5.184%	-1.54%
4	4		C#	4.409%	+0.14%
5	**5**		**Python**	**3.919%**	**-0.34%**
6	7	^	Visual Basic .NET	3.174%	+0.61%
7	6	v	PHP	3.009%	+0.24%
8	8		JavaScript	2.667%	+0.33%
9	11	^	Delphi/Object Pascal	2.544%	+0.54%
10	14	^^	Swift	2.268%	+0.68%
11	9	v	Perl	2.261%	+0.01%
12	10	v	Ruby	2.254%	+0.02%

各个操作系统里的Python：Windows

- 各个版本的Windows都需要额外安装Python（32 / amd64）

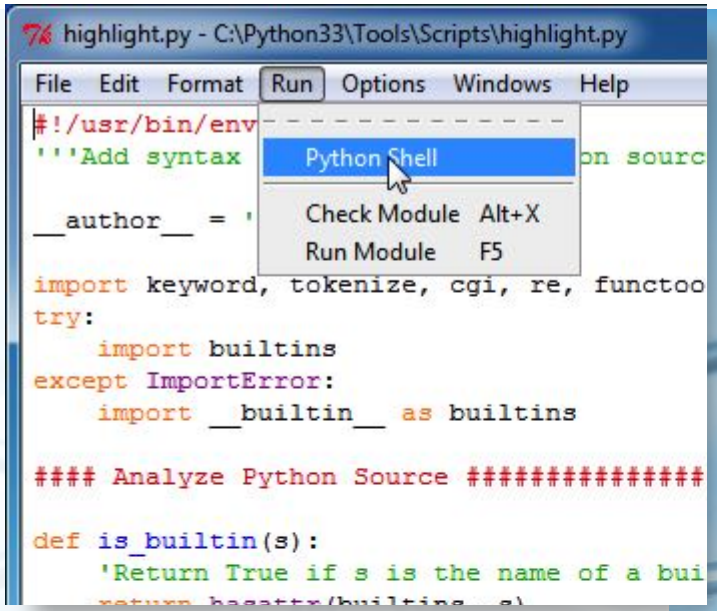
  安装成功完成后，从程序菜单找Python



- Command Line是命令行界面

  只能交互式执行单个语句

  输入quit()来退出Python命令行

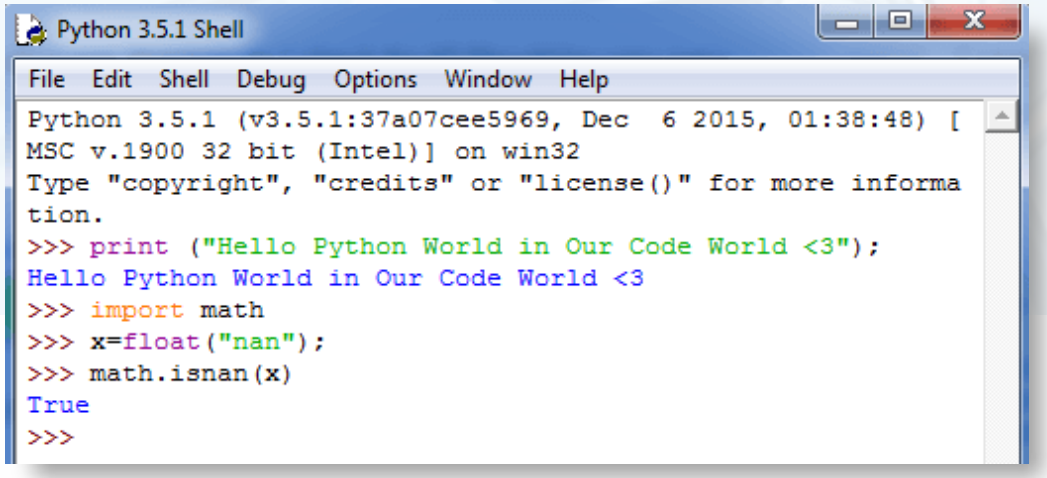


- IDLE是Python的图形界面

  拥有两种窗口：

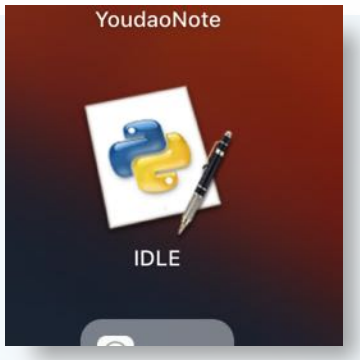
  交互式单句执行窗口：Shell

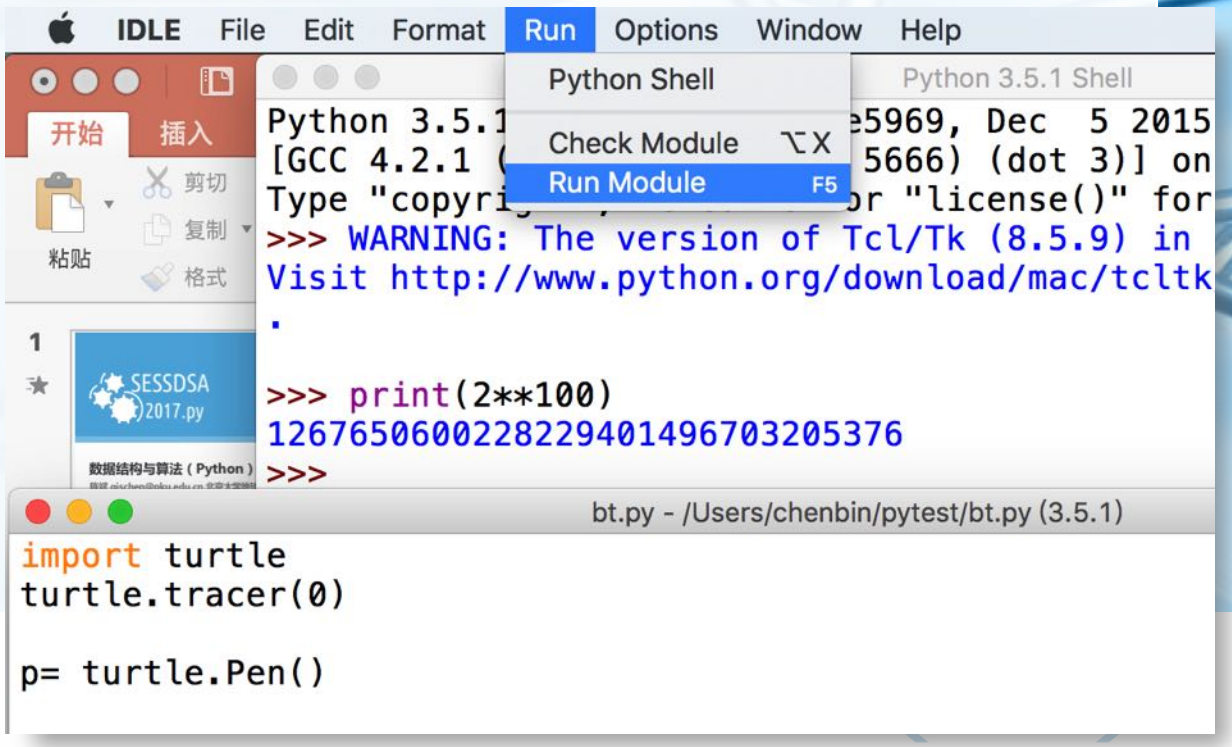
  程序代码文件编辑窗口，编辑和保存程序文件，并在Shell中执行程序（Run->Run Module）



各个操作系统里的Python：macOS

- macOS内置有Python，但可以安装更高版本的Python
- 命令行界面从“Launchpad->其它->终端“，输入python3
- IDLE从"Launchpad"直接运行
- 命令行界面和IDLE跟Windows下一样





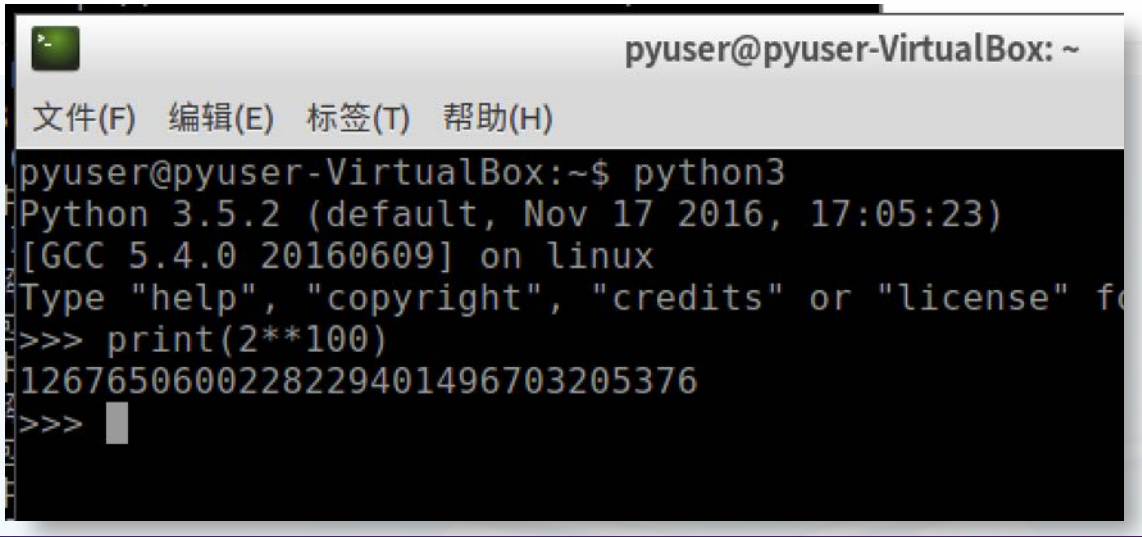
各个操作系统里的Python：各种Linux

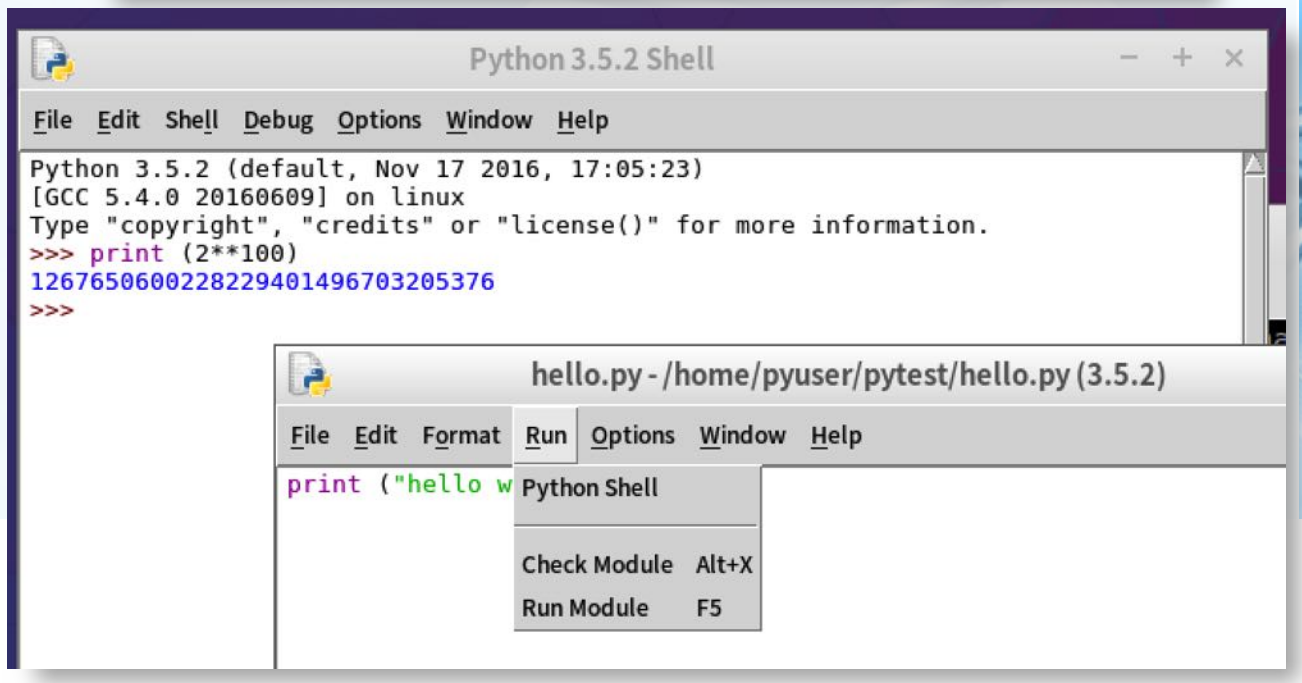
- 各种Linux都内置了Python3
- 命令行界面也是从终端输入python3来启动
- 也具备IDLE的图形界面

  （需要一个简单命令自动安装idle3）

  ```
  sudo apt install idle3
  ```

- 操作也是一样





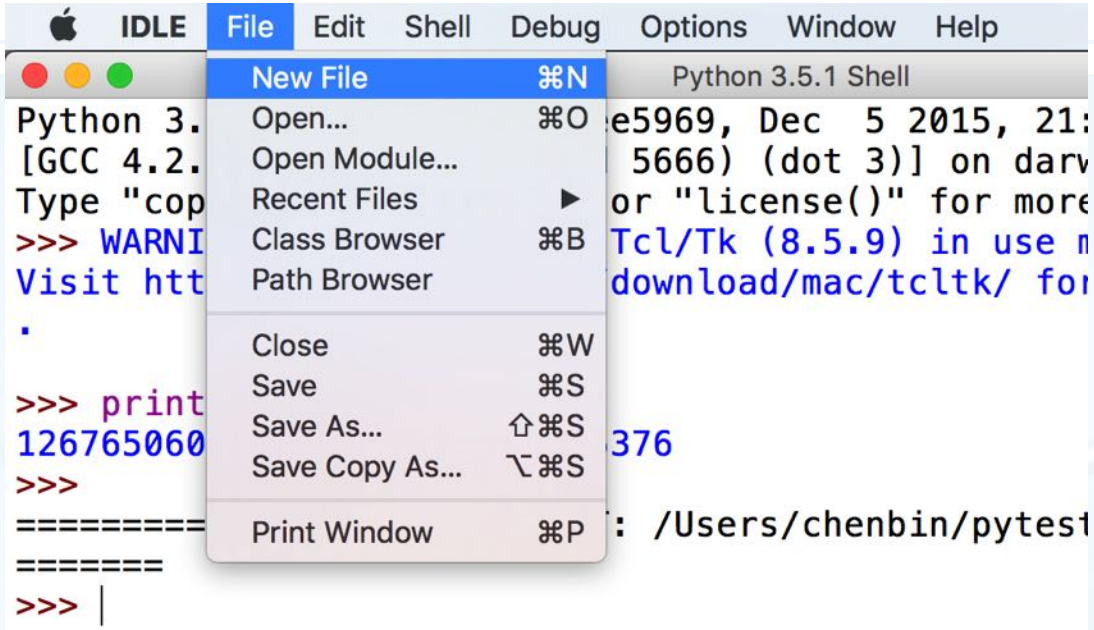
在IDLE中编辑和运行Python程序

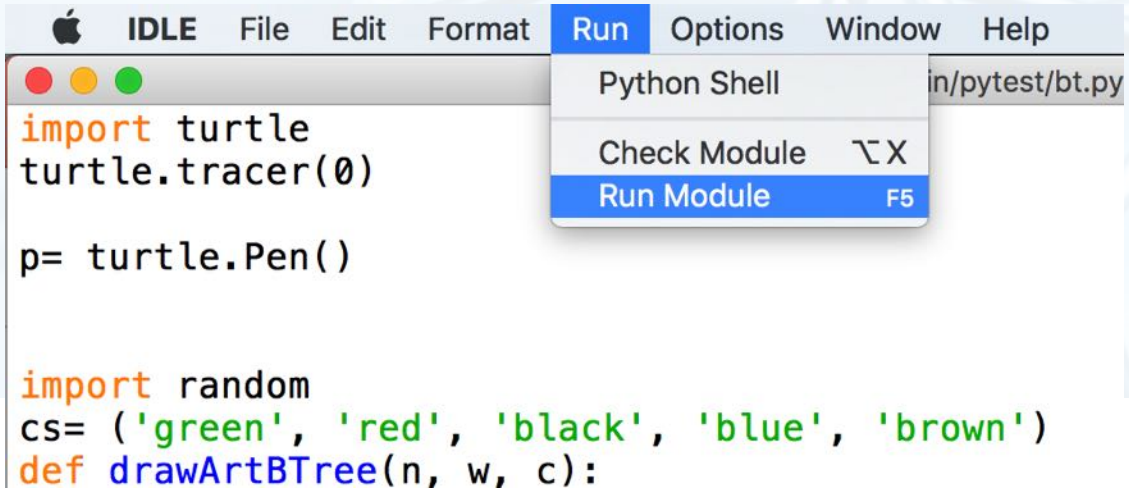
- 启动IDLE
- File->New File
- 在文件编辑窗口中输入代码
- 保存代码文件

  注意保存在“D:\homework”或者“文稿/homework”这样的目录下

  目录名和文件名不要用中文

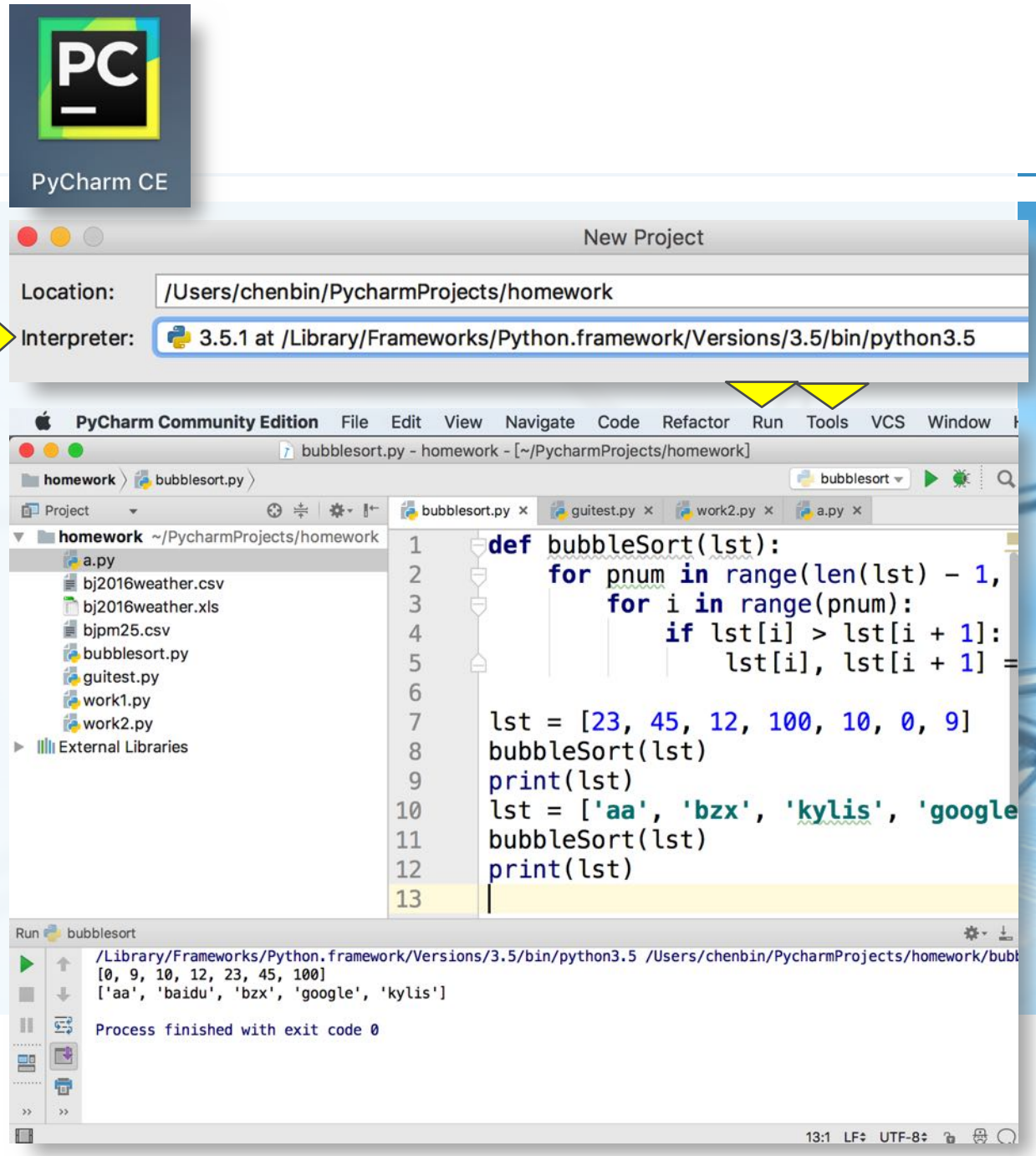
- Run->Run Module运行程序
- 在Python Shell中查看运行结果





集成开发环境：PyCharm

- 首先New Project
- 创建homework目录
- 选择好Python3的解释器
- 然后File->New...来创建Python File
- 有巨多高级特性帮助快速编写程序
- Run->Run...来运行程序
- 可以Tools->Python Console调出命令行界面来执行单条语句

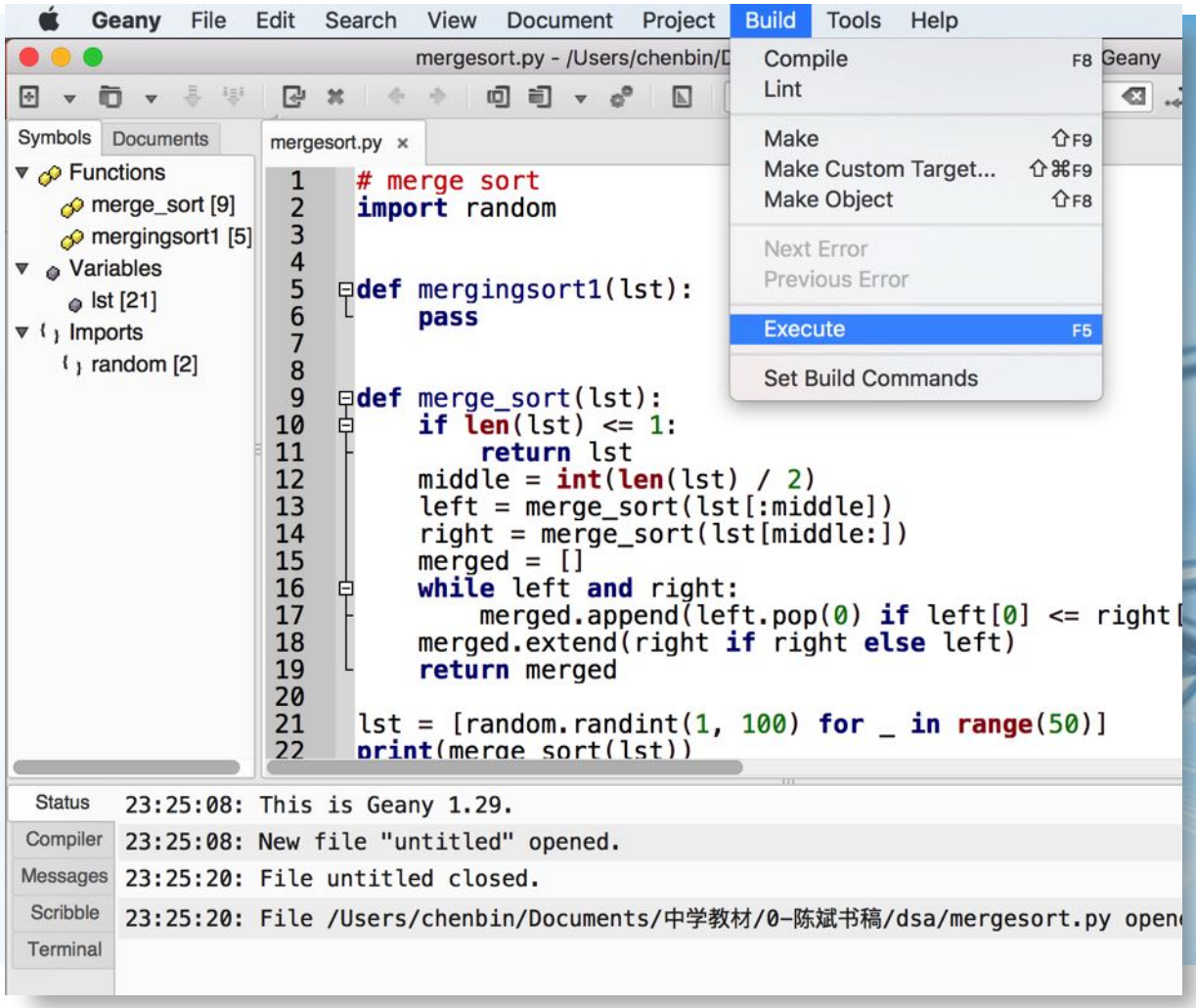


集成开发环境：Geany

- 本质上是一个带执行程序功能的代码编辑器
- 可以编辑执行各种语言的程序

  Python / C / C++ / Java / HTML等等
- 保存程序后，Build->Execute

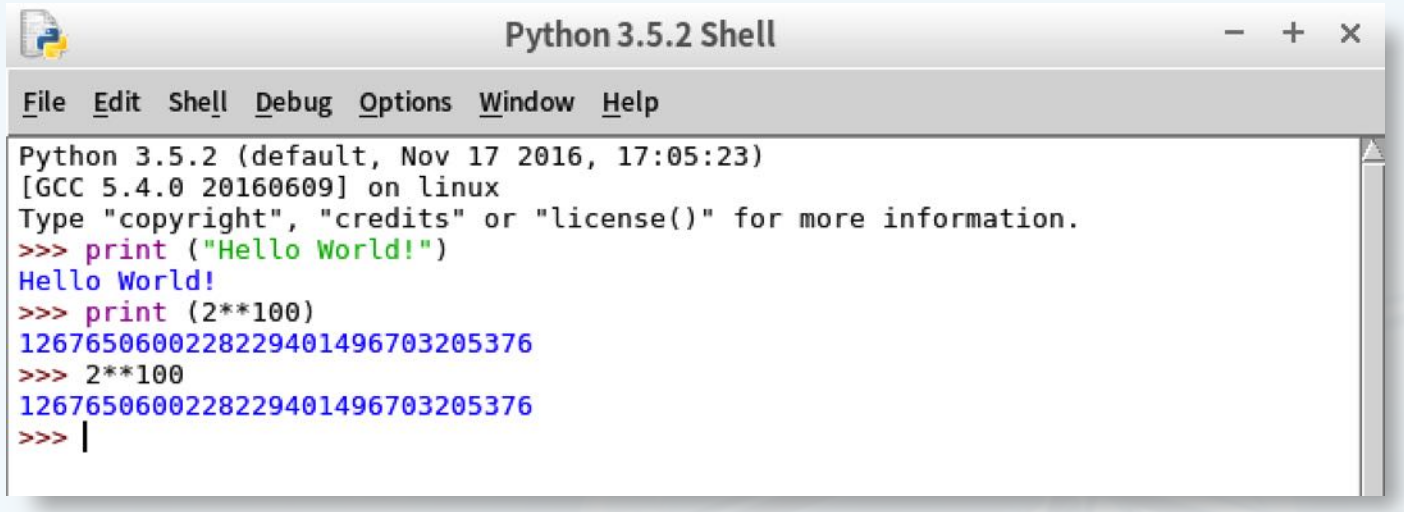
  （可以在Set Build Commands中设置Python解释器的版本）



第一个Python语句：超级计算器

- 打开IDLE
- 在Python Shell中输入语句

```
print ("Hello World!")
```



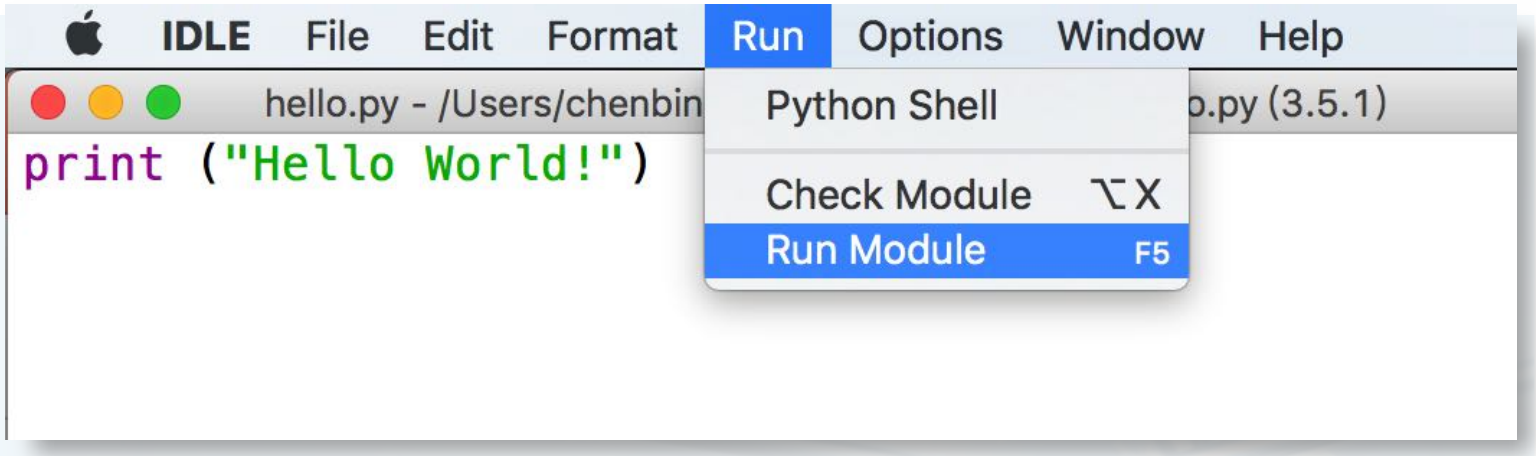
- 立即看到运行结果！
- 可以计算2^{100}！
- 也可以直接输入算式，当计算器用
- 超级大的数都没问题

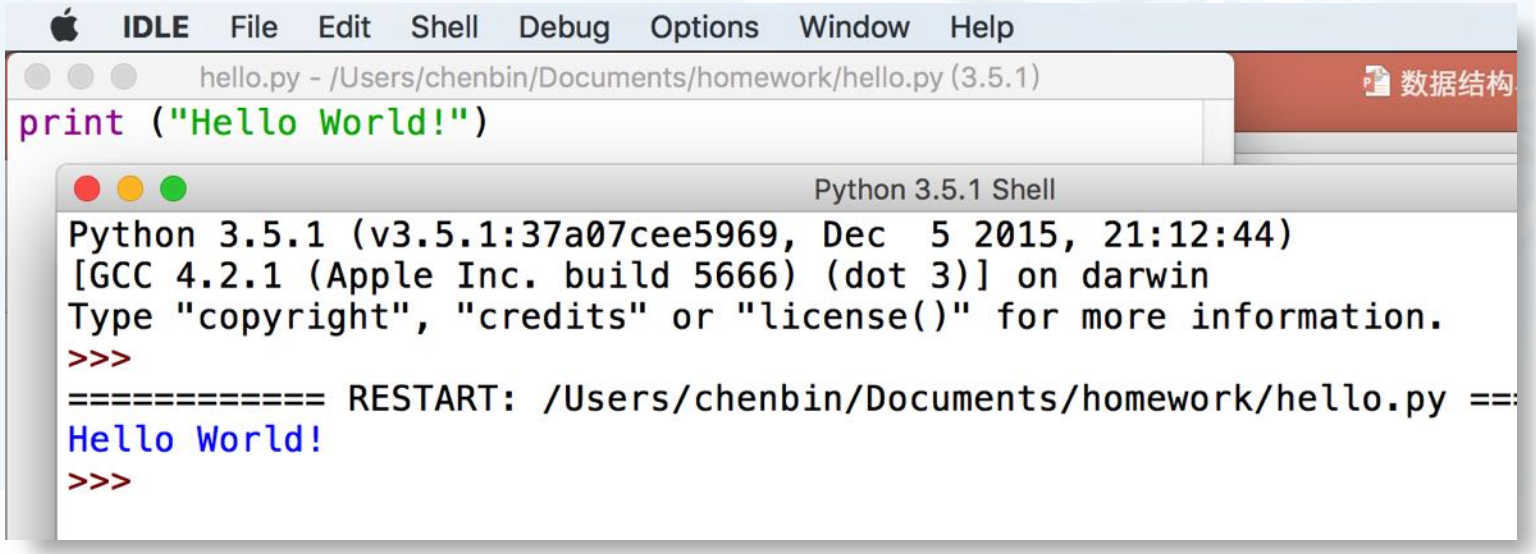
第一个Python程序：Hello World！

- 打开IDLE
- File->New File
- 输入代码

```
print ("Hello World!")
```

- File->Save
- Run->Run Module





Python程序设计哲学：优雅、明确、简单

Python版归并排序

```python
def merge_sort(lst):
    if len(lst) <= 1:
        return lst
    middle = int(len(lst) / 2)
    left = merge_sort(lst[:middle])
    right = merge_sort(lst[middle:])
    merged = []
    while left and right:
        merged.append(left.pop(0) if left[0]
    merged.extend(right if right else left)
    return merged
```

C语言版归并排序

```c
void merge_sort_recursive(int arr[], int reg[], int start, int end) {
    if (start >= end)
        return;
    int len = end - start, mid = (len >> 1) + start;
    int start1 = start, end1 = mid;
    int start2 = mid + 1, end2 = end;
    merge_sort_recursive(arr, reg, start1, end1);
    merge_sort_recursive(arr, reg, start2, end2);
    int k = start;
    while (start1 <= end1 && start2 <= end2)
        reg[k++] = arr[start1] < arr[start2] ? arr[start1++] : arr[sta
    while (start1 <= end1)
        reg[k++] = arr[start1++];
    while (start2 <= end2)
        reg[k++] = arr[start2++];
    for (k = start; k <= end; k++)
        arr[k] = reg[k];
}
void merge_sort(int arr[], const int len) {
    int reg[len];
    merge_sort_recursive(arr, reg, 0, len - 1);
}
```

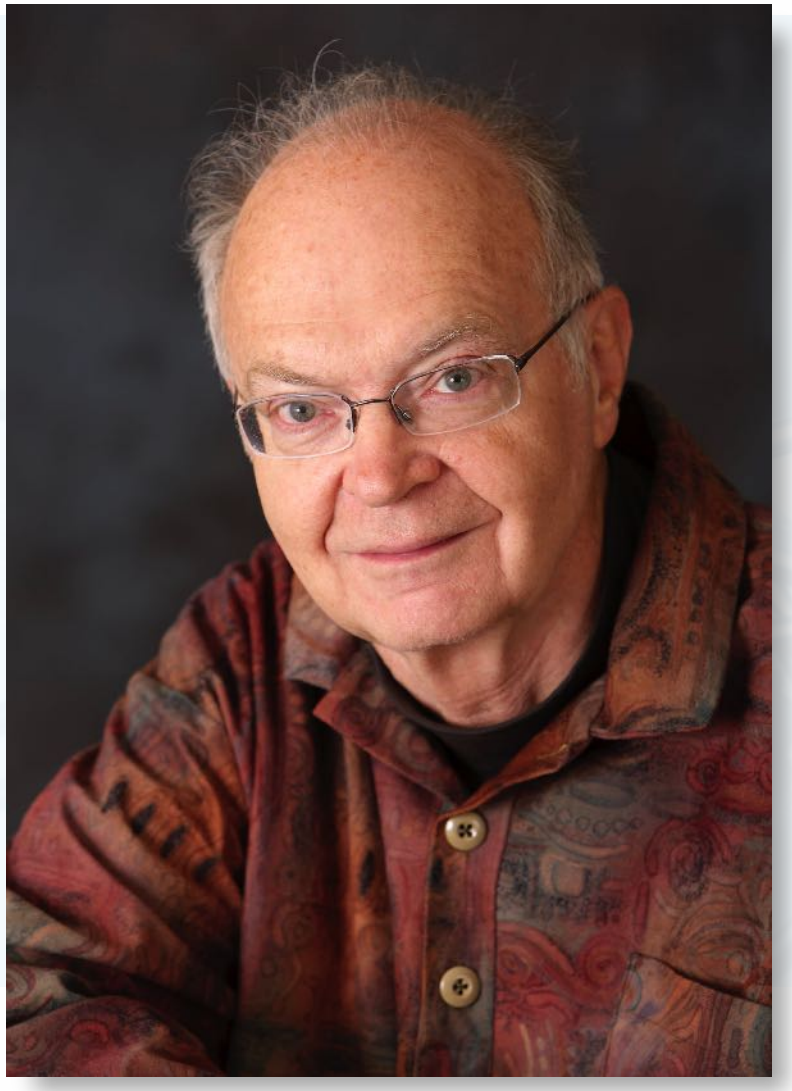
代码强制缩进：视觉效果和功能的统一


```
1  static OSStatus
2  SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedPa
3                                   uint8_t *signature, UInt16 signatureLen)
4  {
5      OSStatus        err;
6      ...
7
8      if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9          goto fail;
10     if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11         goto fail;
12         goto fail;
13     if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14         goto fail;
15     err = sslRawVerify(ctx,
16                        ctx->peerPubKey,
17                        dataToSign,              /* plaintext */
18                        dataToSignLen,           /* plaintext length */
19                        signature,
20                        signatureLen
21     if(err) {
22         sslErrorLog("SSLDecodeSigne
23                     "returned %d\n"
24         goto fail;
25     }
26
27 fail:
28     SSLFreeBuffer(&signedHashes);
29     SSLFreeBuffer(&hashCtx);
30     return err;
31 }
```

```
7
8        if ((err = SSLHashSHA1
9            goto fail;
10       if ((err = SSLHashSHA1
11           goto fail;
12           goto fail;
13       if ((err = SSLHashSHA1
14           goto fail;
15       err = sslRawVerify(ctx
```

程序是写给人读的，只是偶尔让计算机执行一下

- Python的强制缩进规范完成了关键部分
- 我们还需要良好的编程规范
  - 变量、函数、类命名
  - 注释和文档
  - 一些编程设计上的良好风格
- Programs are meant to be read by humans and only incidentally for computers to execute.—Donald Ervin Knuth

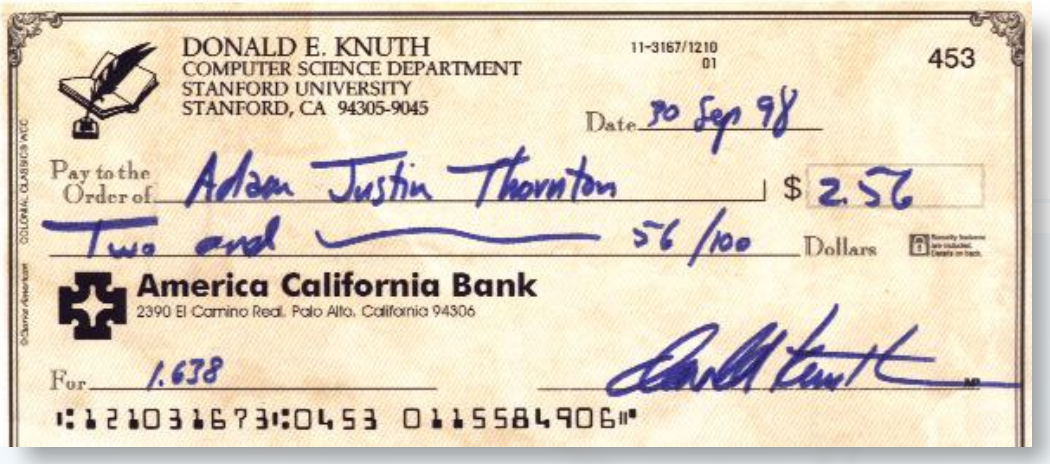


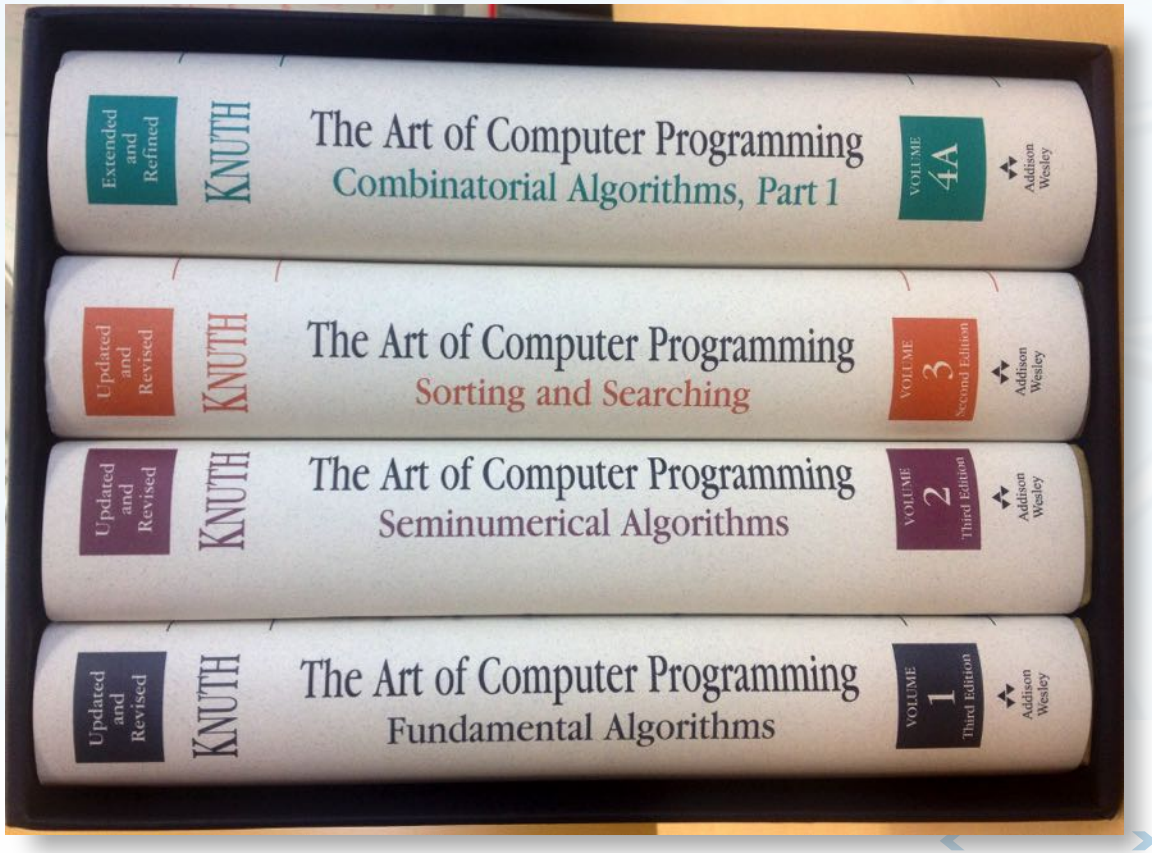
程序员的精彩人生

- 鸿篇巨著《计算机程序设计艺术》
- 为了巨著印刷漂亮，开发了伟大的排版软件T_EX

  有趣的版本号3.14159265，最后是π

  奖励bug提交者，从128美分开始翻倍，但得到奖金的人往往不愿拿支票去兑现
- 字符串快速匹配KMP算法
- 1974年获得图灵奖





Python哲学

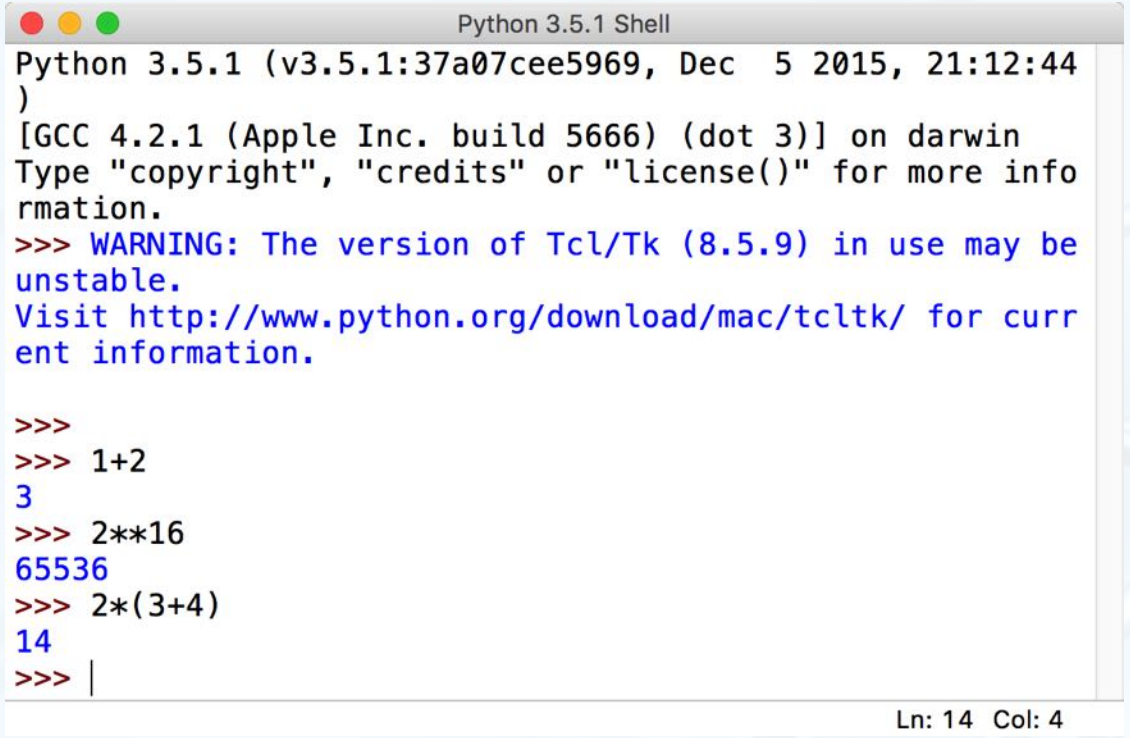

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

上机练习

- **启动Python命令行界面**
  输入算术表达式立即求值

- **启动Python IDLE交互界面**
  输入样例程序，运行通过

  样例程序见课程网站



Python语言的几个要件

数据对象和组织

- 对现实世界实体和概念的抽象
- 分为简单类型和容器类型
- 简单类型用来**表示**值
  整数int、浮点数float、复数complex、逻辑值bool、字符串str
- 容器类型用来**组织**这些值
  列表list、元组tuple、集合set、字典dict
- 数据类型之间几乎都可以转换

赋值和控制流

- 对现实世界处理和过程的抽象
- 分为运算语句和控制流语句
- 运算语句用来实现**处理与暂存**
  表达式计算、函数调用、赋值
- 控制流语句用来**组织**语句描述过程
  顺序、条件分支、循环
- 定义语句也用来组织语句，描述一个包含一系列处理过程的计算单元
  函数定义、类定义

Python数据类型：整数int、浮点数float

- 最大的特点是不限制大小
- 浮点数受到17位有效数字的限制
- 常见的运算包括加、减、乘、除、整除、求余、幂指数等
- 浮点数的操作也差不多
- 一些常用的数学函数如sqrt/sin/cos等都在math模块中

```
import math
math.sqrt(2)
```

```
>>> 5
5
>>> -100
-100
>>> 5 + 8
13
>>> 90 - 10
80
>>> 4 * 7
28
>>> 7 / 2
3.5
>>> 7 // 2
3
>>> 7 % 3
1
>>> 3 ** 4
81
>>> 2 ** 100
1267650600228229401496703205376
>>> divmod(9, 5)
(1, 4)
>>>
```

Python数据类型：复数

- Python内置对复数的计算
- 支持所有常见的复数计算
- 对复数处理的数学函数在模块cmath中

```
import cmath
cmath.sqrt(1+2j)
```

```
>>> 1+3j
(1+3j)
>>> (1+2j)*(2+3j)
(-4+7j)
>>> (1+2j)/(2+3j)
(0.6153846153846154+0.07692307692307691j)
>>> (1+2j)**2
(-3+4j)
>>> (1+2j).imag
2.0
>>> (1+2j).real
1.0
>>>
```

Python数据类型：逻辑值

- 逻辑值仅包括True/False两个
- 用来配合if/while等语句做条件判断
- 其它数据类型可以转换为逻辑值：
  例如数值：0与非0等

```
>>> True
True
>>> False
False
>>> 1>2
False
>>> 23<=34
True
>>> bool(0)
False
>>> bool(999)
True
>>> if (2>1):
        print ("OK")


OK
>>>
```

Python数据类型：字符串

- 最大的特点是Python字符串不可修改，只能生成新的字符串
- 用双引号或者单引号都可以表示字符串
- 多行字符串用三个连续单引号表示
- 特殊字符用转义符号“\”表示

  制表符\t，换行符号\n
- 字符串操作：

  +连接、*复制、len长度

  [start:end:step]用来提取一部分

```
>>> 'abc'
'abc'
>>> "abc"
'abc'
>>> '''abc def
ghi jk'''
'abc def\nghi jk'
>>> "Hello\nWorld!"
'Hello\nWorld!'
>>> print ("Hello\nWorld!")
Hello
World!
>>> 'abc' + 'def'
'abcdef'
>>> 'abc' * 4
'abcabcabcabc'
>>> len('abc')
3
>>> 'abcd'[0:2]
'ab'
>>> 'abcd'[0::2]
'ac'
```

Python数据类型：字符串

- 一些高级操作：

  split：分割；join：合并

  upper/lower/swapcase：大小写相关

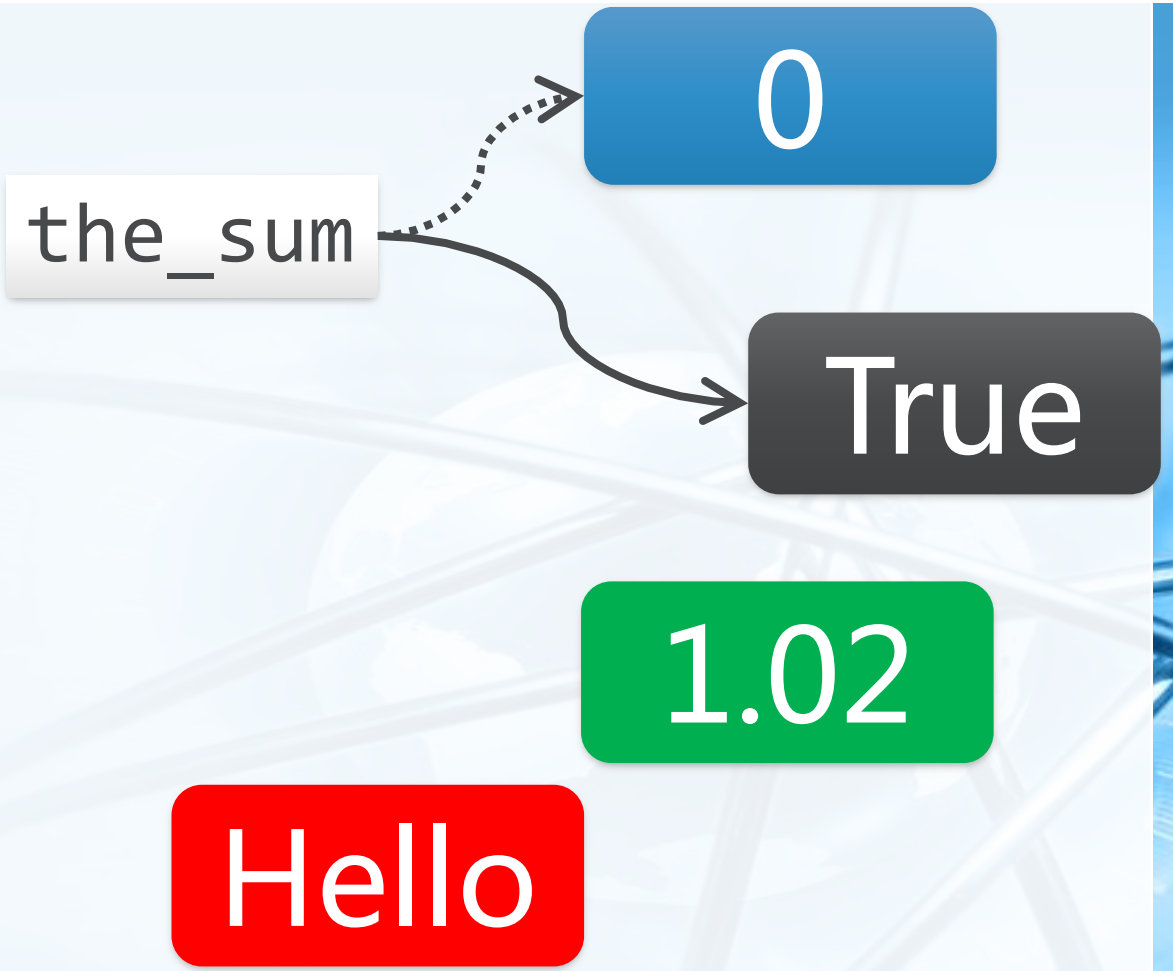
  ljust/center/rjust：排版左中右对齐

  replace：替换子串

```
>>> 'You are my sunshine.'.split(' ')
['You', 'are', 'my', 'sunshine.']
>>> '-'.join(["One", "for", "Two"])
'One-for-Two'
>>> 'abc'.upper()
'ABC'
>>> 'aBC'.lower()
'abc'
>>> 'Abc'.swapcase()
'aBC'
>>> 'Hello World!'.center(20)
'     Hello World!     '
>>> 'Tom smiled, Tom cried, Tom shouted'.replace('Tom', 'Jane')
'Jane smiled, Jane cried, Jane shouted'
```

Python变量机制：引用数据对象

- 赋值语句`the_sum = 0`，实际上是创建了名为the_sum的变量，然后指向数据对象“0”
- 所以变量可以随时指向任何一个数据对象，比如True，1.02，或者"Hello"
- 变量的类型随着指向的数据对象类型改变而改变！



上机练习：基本数据类型

- 数值基本运算：33和7

  +, -, *, /, //, %, **

  hex(), oct(), bin()

- 类型转换

  1, 0, 'abc', None, 1.2, False, ''

  str(), bool(), int(), float()

  is None, ==, !=

- 字符串基本操作

  +, *, len(), [], in

  ord(), chr()

  含有中文的字符串

- 字符串高级操作

  s='abcdefg12345'

  切片：获得defg12，获得fg12345，获得54321， 获得aceg2

  t='Mike and Tom'

  split拆分、

  upper/lower/swapcase修改大小写、

  ljust/center/rjust排版30位宽度左中右对齐

  replace将Mike替换为Jerry

Python容器类型：列表和元组

- Python中有几种类型是一系列元素组成的序列，以整数作为索引
- 字符串str就是一种同类元素的序列
- 列表list和元组tuple则可以容纳不同类型的元素，构成序列
- 元组是不能再更新（不可变）序列

  字符串也是不能再更新的序列

- 列表则可以删除、添加、替换、重排序列中的元素

  可变类型

字符串str

0	1	2	3	4	5	6	7	8
H	*e*	*l*	*l*	*o*		*T*	*o*	*m*

列表list

0	1	2	3	4	5	6	7	8
123	2.4	'ab'	True	None	[1,2]	(2,3)	556	0

元组tuple

0	1	2	3	4	5	6	7	8
123	2.4	'ab'	True	None	[1,2]	(2,3)	556	0

容器类型：列表和元组

- 创建列表：[ ]或者list()
- 创建元组：( )或者tuple()
- 用索引[n]获取元素（列表可变）
- +：连接两个列表 / 元组
- *：复制n次，生成新列表 / 元组
- len()：列表 / 元组中元素的个数
- in：某个元素是否存在
- [start : end : step]：切片

```
>>> []
[]
>>> list()
[]
>>> alist = [1, True, 
>>> alist[0]
1
>>> alist + ["Hello"]
[1, True, 0.234, 'Hell
>>> alist * 2
[1, True, 0.234, 1, Tr
>>> len(alist)
3
>>> 1 in alist
True
>>> alist
[1, True, 0.234]
>>> alist[1:3]
[True, 0.234]
>>> alist[0:3:2]
[1, 0.234]
>>> alist[::-1]
[0.234, True, 1]
```

```
>>> ()
()
>>> tuple()
()
>>> atuple = (1, True, 0.234)
>>> atuple[0]
1
>>> atuple + ("Hello",)
(1, True, 0.234, 'Hello')
>>> atuple * 2
(1, True, 0.234, 1, True, 0.234)
>>> len(atuple)
3
>>> 1 in atuple
True
>>> atuple
(1, True, 0.234)
>>> atuple[1:3]
(True, 0.234)
>>> atuple[0:3:2]
(1, 0.234)
>>> atuple[::-1]
(0.234, True, 1)
```

```
>>> alist[0] = False
>>> alist
[False, True, 0.234]
```

```
>>> atuple[0] = False
Traceback (most recent call last):
  File "<pyshell#93>", line 1, in <module>
    atuple[0] = False
TypeError: 'tuple' object does not support item assignment
```

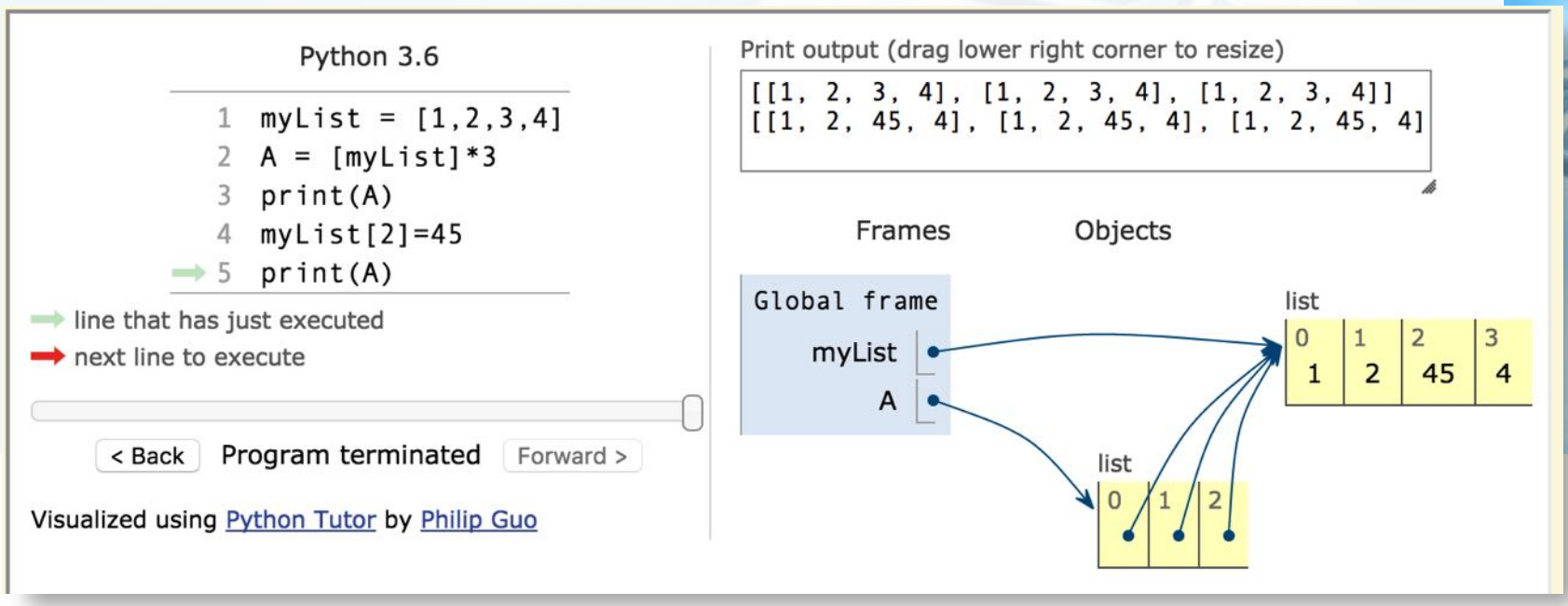
列表list的其它方法

方法名称	使用例子	说明
append	alist.append(item)	列表末尾添加元素
insert	alist.insert(i,item)	列表中i位置插入元素
pop	alist.pop()	删除最后一个元素，并返回其值
pop	alist.pop(i)	删除第i个元素，并返回其值
sort	alist.sort()	将表中元素排序
reverse	alist.reverse()	将表中元素反向排列
del	del alist[i]	删除第i个元素
index	alist.index(item)	找到item的首次出现位置
count	alist.count(item)	返回item在列表中出现的次数
remove	alist.remove(item)	将item的首次出现删除

可变类型的变量引用情况

- 由于变量的引用特性
- 可变类型的变量操作需要注意
- 多个变量通过赋值引用同一个可变类型对象时
- 通过其中任何一个变量改变了可变类型对象
- 其它变量也看到了改变

```
alist = [1,2,3,4]
blist = alist
blist[0] = 'abc'
clist = alist[:]
Clist[0] = None
```



常用的连续序列生成器：range函数

- range(n)
  从0到n-1的序列
- range(start, end)
  从start到end-1的序列
- range(start, end, step)
  从start到end-1，步长间隔step
  step可以是负数
- range函数返回range类型的对象，可以直接当做序列用，也可以转换为list或者tuple等容器类型

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
>>> list(range(10, 1, -2))
[10, 8, 6, 4, 2]
>>> range(10)
range(0, 10)
>>> tuple(range(10))
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Python容器类型：集合set

- 集合是不重复元素的无序组合
- 用set()创建空集
- 可用set()从其它序列转换生成集合
- 集合的常见操作

  in: 判断元素是否属于集合

  |, union(): 并集

  &, intersection() : 交集

  -, difference() : 差集

  ^, symmetric_difference() : 异或

  <=,<,>=,>: 子集/真子集/超集/真超集

```
>>> set()
set()
>>> aset = set('abc')
>>> aset
{'c', 'a', 'b'}
>>> 'a' in aset
True
>>> aset | set('bcd')
{'c', 'd', 'a', 'b'}
>>> aset & set(['b', 'c', 'd'])
{'c', 'b'}
>>> aset - set(('b', 'c', 'd'))
{'a'}
>>> aset ^ set('bcd')
{'a', 'd'}
>>> aset <= set('abcd')
True
>>> aset > set('abcd')
False
```

Python容器类型：集合set

- add(x)：集合中添加元素
- remove(x)：集合中删除指定元素
- pop()：删除集合中任意元素并返回其值
- clear()：清空集合成为空集
- 如果经常需要判断元素是否在一组数据中，这些数据的次序不重要的话，推荐使用集合，可以获得比列表更好的性能

```
>>> aset
{'c', 'a', 'b'}
>>> aset.add(1.23)
>>> aset
{'c', 1.23, 'a', 'b'}
>>> aset.remove('b')
>>> aset
{'c', 1.23, 'a'}
>>> aset.pop()
'c'
>>> aset
{1.23, 'a'}
>>> aset.clear()
>>> aset
set()
```

Python容器类型：字典dict

- 字典是通过键值key来索引元素value，而不是象列表是通过连续的整数来索引
- 字典是可变类型，可以添加、删除、替换元素
- 字典中的元素value没有顺序，可以是任意类型
- 字典中的键值key可以是任何不可变类型（数值 / 字符串 / 元组）

```
>>> student = {'name':'Tom', 'age':20,
               'gender':'Male', 'course':['math', 'computer']}
>>> student
{'name': 'Tom', 'age': 20, 'course': ['math', 'computer'], '
gender': 'Male'}
>>> student['name']
'Tom'
>>> student['age']
20
>>> student['age'] = 19
>>> student
{'name': 'Tom', 'age': 19, 'course': ['math', 'computer'], '
gender': 'Male'}
>>> student['course'].append('chemistry')
>>> student
{'name': 'Tom', 'age': 19, 'course': ['math', 'computer', 'c
hemistry'], 'gender': 'Male'}
>>> 'gender' in student
True
>>> student.keys()
dict_keys(['name', 'age', 'course', 'gender'])
>>> student.values()
dict_values(['Tom', 19, ['math', 'computer', 'chemistry'], '
Male'])
>>> student.items()
dict_items([('name', 'Tom'), ('age', 19), ('course', ['math'
, 'computer', 'chemistry']), ('gender', 'Male')])
```

建立大型数据结构

- **嵌套列表**

  列表的元素是一些列表

  ```
  alist[i][j]
  ```

- **字典的元素可以是任意类型，甚至也可以是字典**

  ```
  bands={'Marxes':['Moe','Curly']}
  ```

- **字典的键值可以是任意不可变类型，例如用元组来作为坐标，索引元素**

  ```
  poi={(100,100):'bus stop'}
  ```

```
>>> alist=[ [23, 34, 45], [True, 'ab']]
>>> alist[0][2]
45
>>> bands={'Marxes':['Moe','Curly'], 'KK':[True, 'moon']}
>>> bands['KK'][0]
True
>>> poi={(100,100):'Zhongguancun', (123,23):'Pizza'}
>>> poi[(100,100)]
'Zhongguancun'
```

输入和输出：input/print函数

- **input(prompt)**

  显示提示信息prompt，用户输入的内容以字符串形式返回

- **print(v1, v2, v3, ...)**

  打印各变量的值输出

  可以带参数end='\n'，缺省为换行，表示打印后以这个字符串结尾

  带参数sep=' '，缺省是空格，表示变量之间用什么字符串隔开

- **格式化字符串**

  '%d %s' % (v1, v2)

```
>>> yname = input ("Please input your name")
Please input your nameTom Hanks
>>> yname
'Tom Hanks'
>>> print (1, 23, 'Hello')
1 23 Hello
>>> print (1, 23, 'Hello', end='')
1 23 Hello
>>> print (1, 23, 'Hello', sep=',')
1,23,Hello
>>> '%d %s' % (23, 'Hello')
'23 Hello'
>>> '%d' % (23,)
'23'
>>> '(%4d):K:%s' % (12, 'Hello')
'(  12):K:Hello'
>>> '(%04d):K:%10s' % (12, 'Hello')
'(0012):K:     Hello'
```

上机练习

- **列表、元组基本操作**

  +, *, len(), [], in

- **列表、元组高级操作**

  mylist=[1,2,3,4,5]

  切片：获得[2,3,4]，获得[3,4,5]，获得[3,2,1]， 获得[1,3,5]

  mytpl=(1,2,3,4,5)同上操作

  t='Mike and Tom'

  split拆分、join合成为'Mike/and/Tom'

- **集合基本操作**

  a=set([1,2,3,4,5])

  b=set([2,4,6,8,10])

  并、交、差、异或、子集

  添加、删除、是否空集

- **字典基本操作**

  mydict={1:Mon, 'line1':3332}

  添加、删除、是否空字典

  取字典所有的key / value

  判断key是否存在