python概述

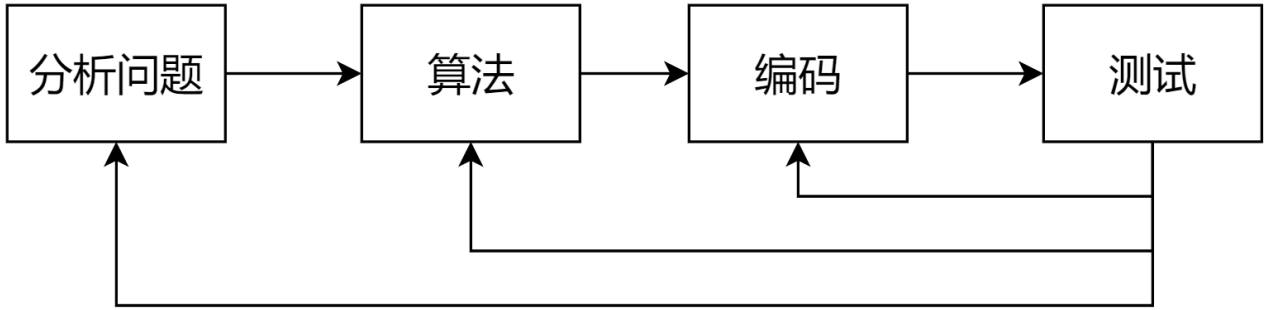
主要内容

- 编程（补充的内容）和python语言概述
- 对象和赋值语句
- 数字类型和算术运算符
- 字符串和相关运算符

编程(Programming)

- 程序(program): 计算机可以执行的一系列指令(instruction)或代码(code)
  - 程序执行：执行程序中包含的指令
- 程序语言：描述这些指令的语言，是“程序员”与“机器”对话的语言
  - 语法（syntax）: 哪些符号或文字的组合方式是正确的
  - 语义（semantics）: 描述程序的意义，当代码运行时计算机干什么
- 算法(algorithm)
  - 描述了为了解决某个问题需要采取的动作以及这些动作的顺序
  - 自然语言、伪代码（pseudo-code, 自然语言混杂着一些程序代码)、程序流程图的形式
- 编程(programming)：为了解决某个问题，设计算法，然后将其转换为采用某种程序语言描述的程序

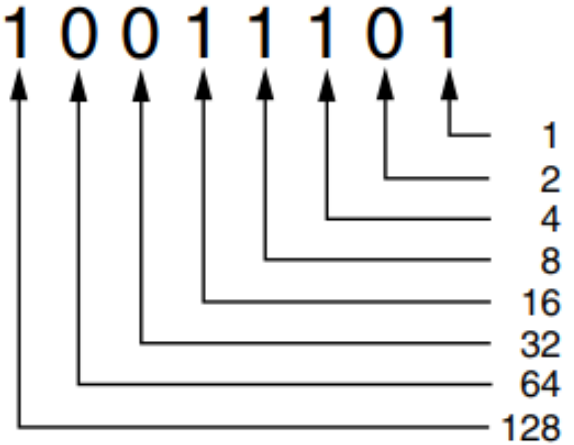
程序语言



程序语言

进制之间的转换：二进制的10011101，对应十进制的

$1*2^7+1*2^4+1*2^3+1*2^2+1*2^0=128 + 16 + 8 + 4 + 1 = 157$



- 计算机的运算采用二进制
  - 所有程序和数据以二进制方式存储
  - 位/比特(bit): 单个二进制数字
  - 字节(byte)：8个连续的比特
- 低级程序语言：
  - 机器语言（Machine code）：
    - 以数字的形式（十六进制）描述了CPU所执行的指令，比如 05 01 23
    - 编写的代码称为目标代码（机器代码）
  - 汇编语言（Assembly Language）：以人可读的方式描述机器代码

    add EAX, 1

    mov [ESP+4], EAX
  - 不同架构的CPU有不同的机器代码：Intel（32和64比特）、ARM、PowerPC等
  - 经常需要大量的指令才能完成某个功能
  - 难学、容易出错

高级程序语言

- 类自然语言，不再依赖某种特定的机器或环境。在不同的平台上会被转换成不同的机器语言
- **语句(statement)**：相当于指令，由关键字、运算符、标点符号以及其他编程元素（字面量等）按照相应的顺序组成
- 按照高级语言编写的程序称为源程序或源代码（source code)
- 更加容易学习和使用，一个语句经常对应着许多条机器代码
- Fortran, C, C++, C#, Java, Javascript, Python, Visual Basic, Ruby, Go等

```
total = price * (tax + 100) / 100
```

```
mov EAX, EBP[-2]
mov EBX, EBP[-4]
add EBX, 100
mul EAX, EBX
div EAX, 100
mov EBP[2], EAX
```

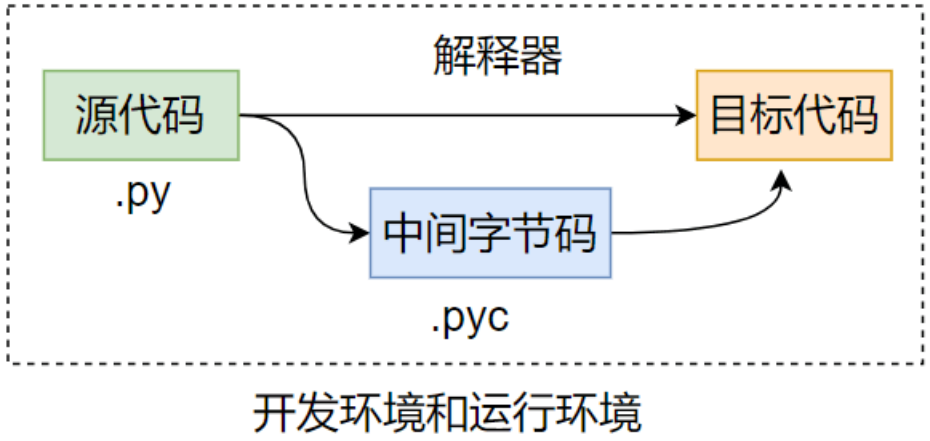
解释和编译

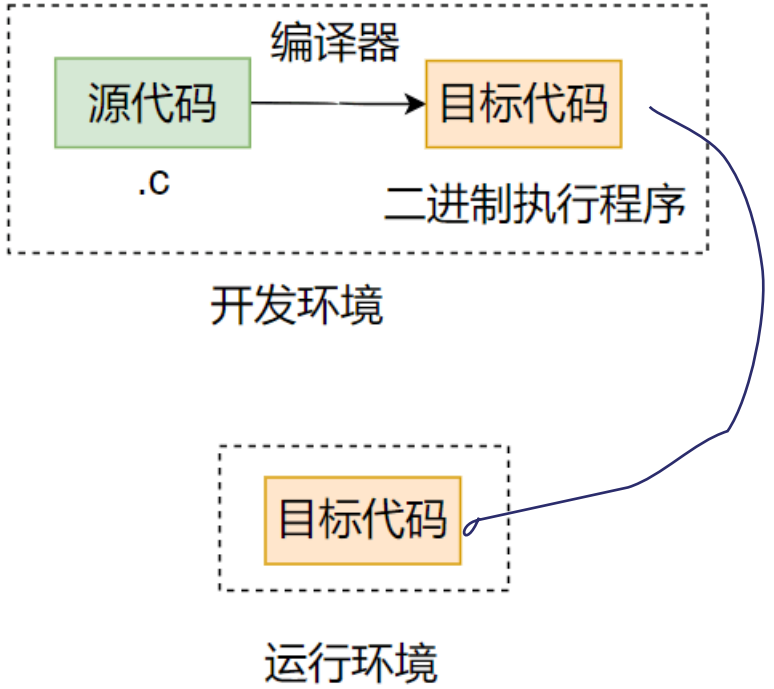
如何将使用**高级语言编写的程序（源代码/源程序source code）**转换为**目标代码（机器码）**？

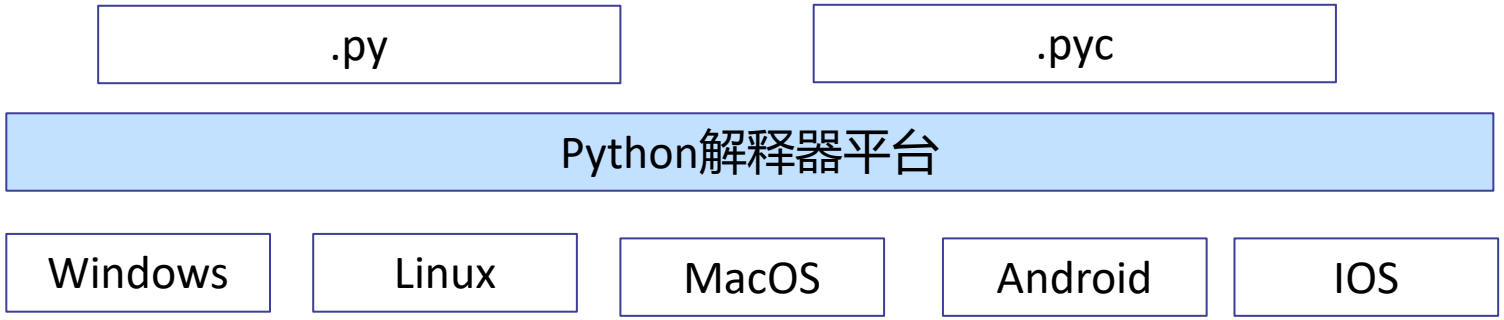
- **解释器（Interpreter）**：将语句翻译成机器码，**并且执行**，Python、Javascript、Perl、PhP等
  - 修改程序方便，修改代码重新运行就可以了
  - 必须有解释器才可以运行，跨平台（只要有解释器，就可以执行）
  - 每次运行，都要进行翻译，运行速度会有影响
- **编译器（Compiler）**：将语句翻译成机器码，**形成目标代码文件**(机器语言），C、C++、VB等
  - 修改程序后需要进行编译
  - 编译一次，编译后的目标代码可以直接在相应的操作系统中运行，不再需要编译器
  - 编译时相比解释可以作更多的优化
- 有些语言将解释和编译结合在一起
  - Java语言：首先通过编译器(javac)将源代码转换为中间的Java字节码（Byte Code），然后在目标机器上通过解释器（java虚拟机）来运行
  - Python语言：解释器首先将.py程序转换为.pyc字节码，然后由解释器解释执行
    - **被import的模块源程序**，还会保存之前转换的.pyc字节码，源程序没有修改时可省略转换过程，优化程序和提高执行速度

解释和编译

```
$ cat hello.py
print('hello world!' )
$ python hello.py
hello world!
```



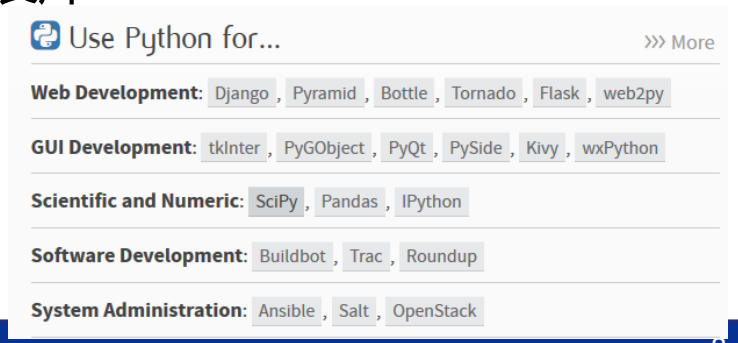





```
$ cat hello.c
#include <stdio.h>

int main(char *argv[])
{
    printf("Hello world!\n");
}
$ gcc hello.c
$ ./a.out
hello world!
```

Python语言

- 1989年吉多.范罗苏姆Guido van Rossum发明Python [`paɪθən]语言
- **简单、易学**：Python是一种代表简单主义思想的语言
- **开源、免费**： Python是FLOSS（Free/Libre and Open Source Software，自由/开放源码软件）之一。使用者可以自由地发布这个软件的拷贝、阅读它的源代码、对它做改动、把它的一部分用于新的自由软件中
- **跨平台**：所编写程序在解释器支持下可无需修改在Windows、 Linux、 Mac等操作系统上使用
- **灵活性**：Python支持多种编程范式，包括过程式编程、面向对象编程、函数式编程
- **可扩展和可嵌入性**：胶水语言，支持采用C、C++等语言编写扩充模块，也可为C、C++程序提供脚本功能
- **丰富的扩展库支持**：拥有大量的几乎支持所有领域应用开发的成熟扩展库
- 可广泛应用在Web应用、图形用户界面开发、系统管理、科学计算、人工智能、大数据、游戏等



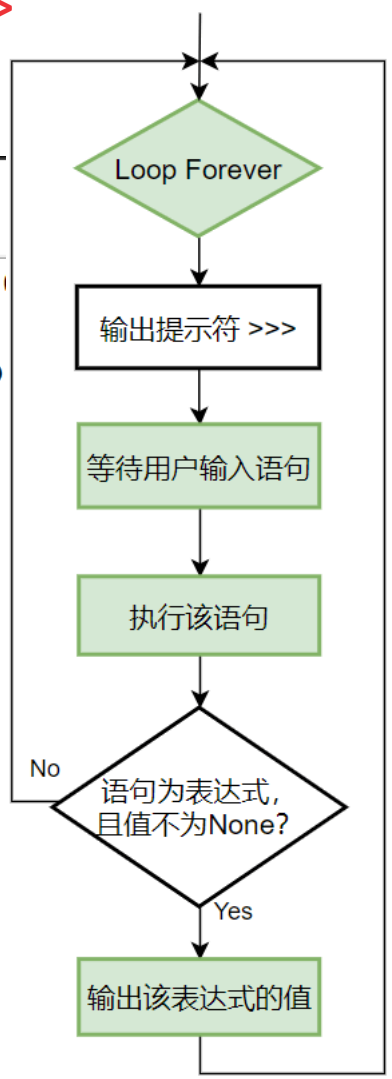
Python版本

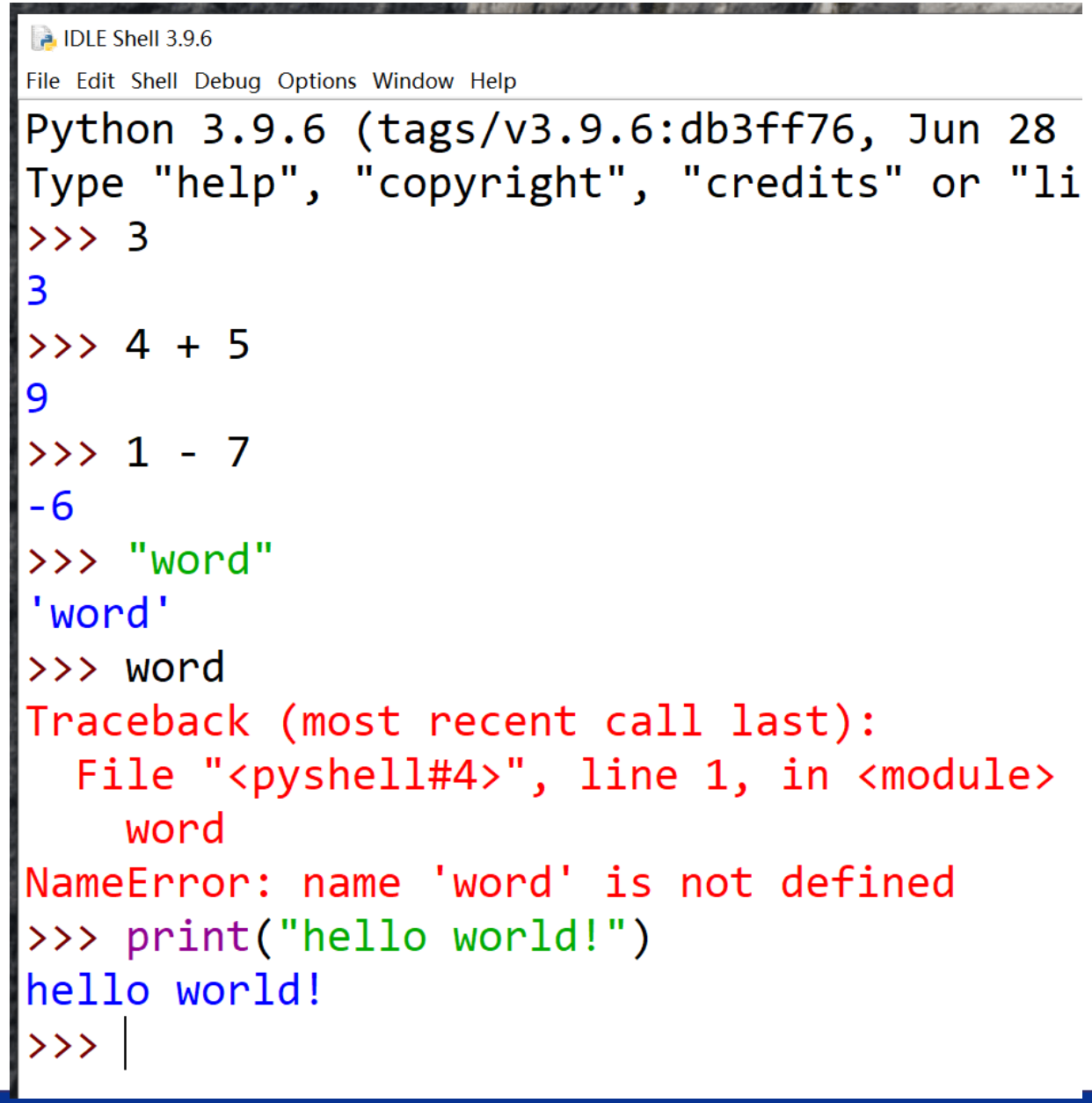
- Python3：我们的教学采用Python3
  - 最初于2008年12月发布。最新版本为2023年2月8发布的3.11.2
  - 3.x版本与2.x版本并不完全兼容
- 可同时安装多个版本，通过菜单选择不同的Python版本。 http://www.python.org/downloads/
- 安装的python平台包括了：
  - python解释器(interpreter)
  - IDLE：集成开发环境IDE（Integrated Development Environment）
    - 将编辑器、调试器、解释器环境综合在一起
    - 其他常用的IDE还包括PyCharm、 Visual Studio Code、 Spyder、 Jupyter等

版本	状态	最初发行日期	结束支持时间	发行计划
3.11	Bugfix	2022-10-24	2027-10	PEP 664
3.10	bugfix	2021-10-04	2026-10	PEP 619
3.9	security	2020-10-05	2025-10	PEP 596
3.8	security	2019-10-14	2024-10	PEP 569
3.7	security	2018-06-27	2023-06-27	PEP 537
2.7	不再支持	2010-07-03	2020-01-01	PEP 373

执行python代码的两种模式：交互式和脚本模式

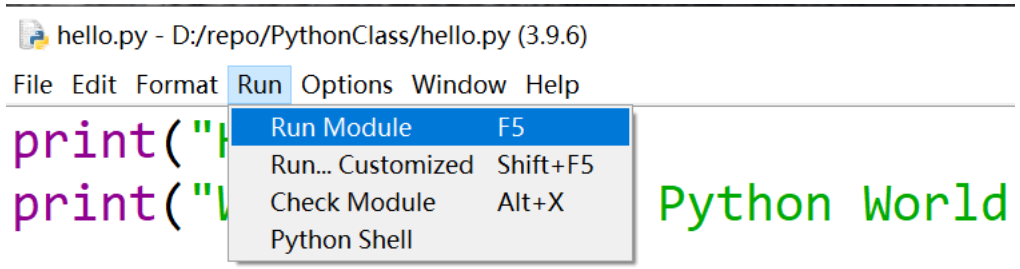
- 交互式：**Read-Evaluate-Print Loop (REPL)**
  - **Python解释器的提示符为 >>>**
  - 可以当成计算器来使用

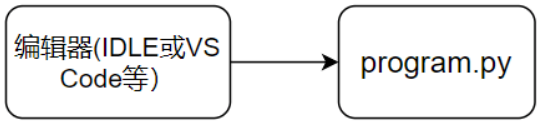


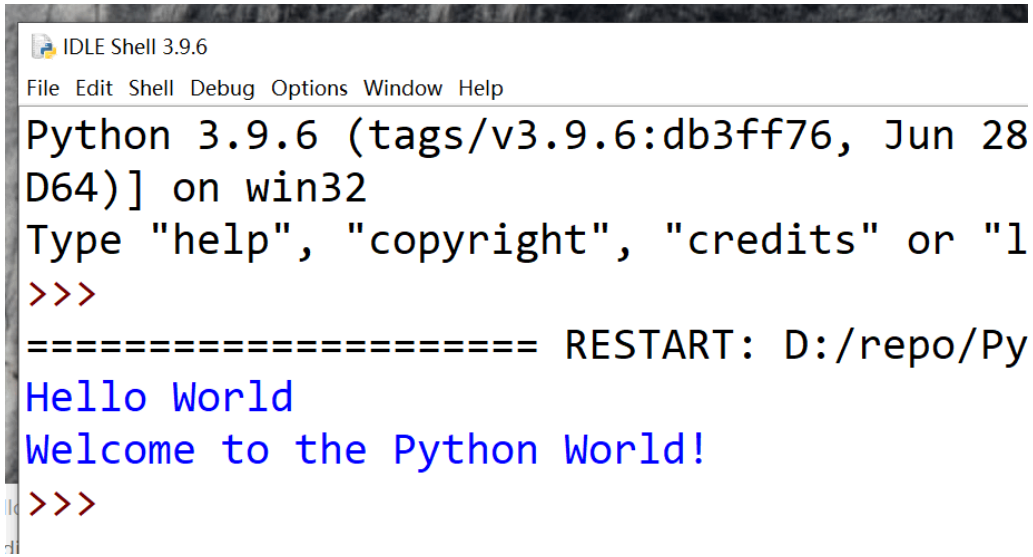


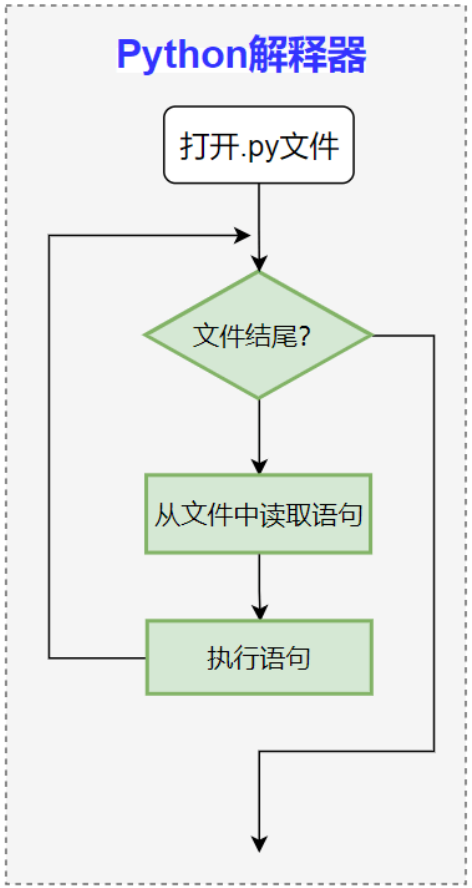
执行python代码的两种模式：交互式和脚本模式

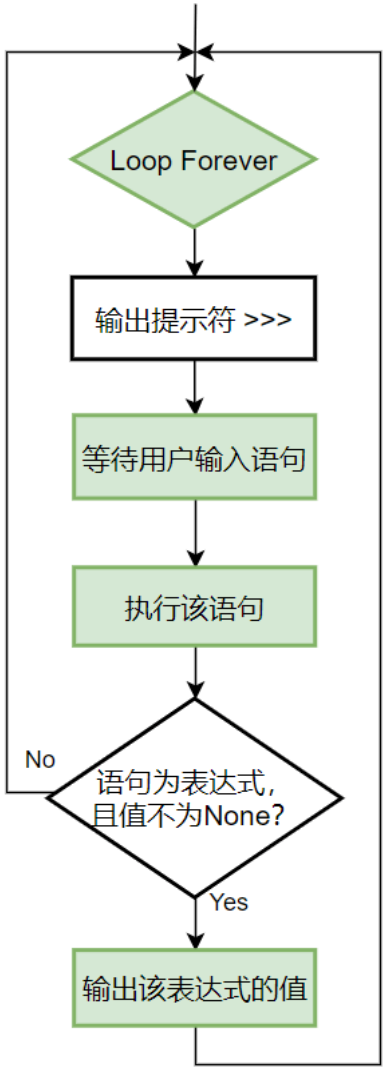
- 脚本(script)方式：
  - 首先通过编辑器或集成开发环境编写python源程序(也称为python脚本），该源程序包括多条python语句
  - **Run子菜单→Run Module(F5)运行**，可以在Python Shell窗口中看到运行结果
    - Read/Evaluate: 解释器读取python源程序的内容，执行这些python语句











内置函数print

- **print(...)** 调用print函数，将相应的值输出到屏幕上：
  - 函数调用：func_name(...)
    - print为要调用的函数名
    - 括号内为调用该函数所传递的参数，多个参数时，参数之间以逗号隔开。print函数支持可变长的参数
  - 功能：将各个参数的值输出到控制台，参数之间以空格隔开，最后输出换行符。**返回None**

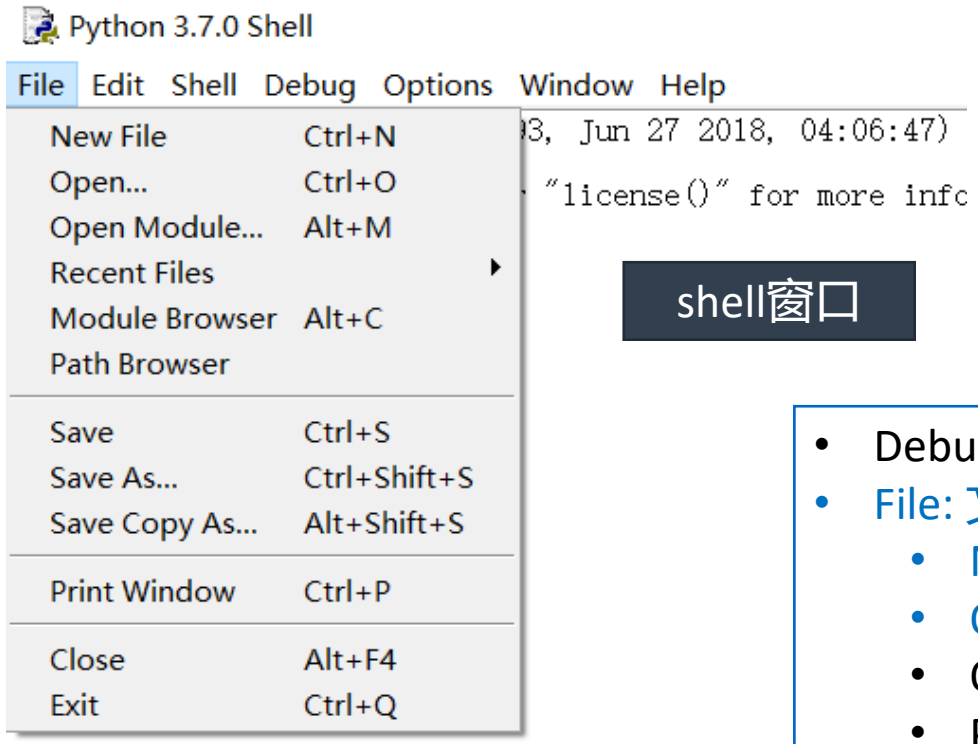
```
>>> print("hello")
hello
>>> print("hello", "world")
hello world
>>> print("hello", "world", "!")
hello world !
>>> print()

>>>
```

- 数学函数：定义域中的元素按照确定的法则映射到值域中的元素
- 编程语言中的函数：接收相应的参数，通过**执行一系列的语句来完成相应的功能**，最后再返回结果

IDLE使用：File菜单

注意：shell窗口的Save/Save As与编辑器窗口的Save/Save As的区别

- Shell窗口和编辑器窗口都有 File/Edit/Options/Window/Help菜单
- 注意File菜单中Save等菜单的含义



shell窗口

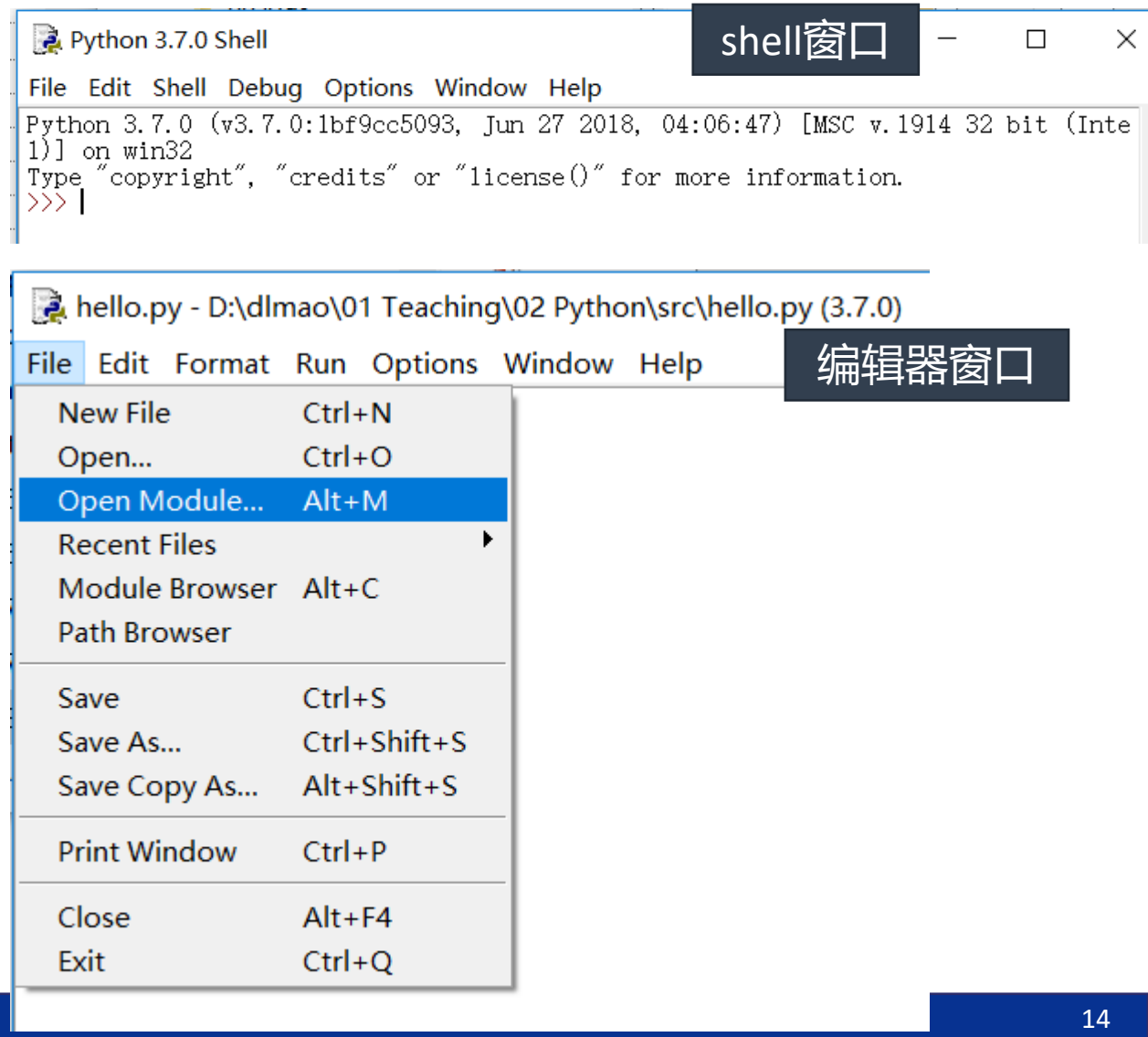
编辑器窗口

- 选择菜单项：**File→New File**（Ctrl-N)创建一个新文件，弹出新的编辑器窗口
- 打开已有的源程序
  - 菜单项**File→Open(Ctrl-O)**，或者
  - 在windows系统中**资源管理器中鼠标右键弹出上下文菜单**，选择Edit With IDLE

- Debug: 调试器相关的操作
- File: 文件相关的操作：
  - New File: 打开编辑器编辑新文件
  - Open: 打开编辑器编辑已有的文件
  - Open Module/ Module Browser : 打开某个模块的源程序
  - Path Browser: 查看sys.path列出的目录，浏览目录并选择相应源程序打开
  - Save: 保存当前窗口(控制台或当前编辑文本)中的文本
  - Save As: 保存当前窗口(控制台或当前编辑文本)中的文本到另一文件名

IDLE： 编写Python源程序(后缀名一般为.py)

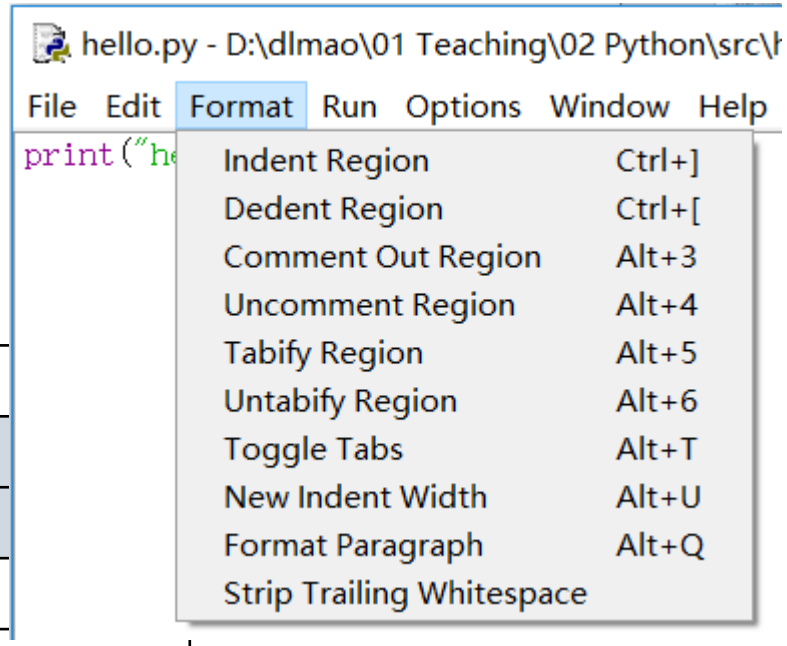
- 选择菜单项：**File→New File**（Ctrl-N)创建一个新文件，弹出新的编辑器窗口，
- 打开已有的源程序
  - 菜单项**File→Open(Ctrl-O)**，或者
  - 在windows系统中**资源管理器中鼠标右键弹出上下文菜单**，选择Edit With IDLE
- **编辑器窗口**：
  - 在代码输入完毕后，**保存该文件**(File→Save, Ctrl-S)，选择要保存在哪个目录，指定该源程序的文件名
  - Run子菜单:
    - Run→Python Shell: 切换到Shell窗口
    - **Run→Run Module(F5)运行**，可以在Python Shell窗口中看到运行结果
    - Run→Run Customized，类似于Run Module，可以指定额外的参数
    - Run→Check Module仅仅检查编写的程序是否有语法错误，并不运行



IDLE快捷键

- **撤销（Ctrl+Z）、全选（Ctrl+A）、复制（Ctrl+C）、粘贴（Ctrl+V）、剪切（Ctrl+X）**

快捷键	功能说明
交互模式：Alt+p	浏览历史命令（上一条）
交互模式：Alt+n	浏览历史命令（下一条）
Ctrl+F6	重启Shell，之前定义的对象和导入的模块全部失效
F1	打开Python帮助文档
Alt+/	（Expand word）自动补全前面曾经出现过的单词，如果之前有多个单词具有相同前缀，则在多个单词中循环选择
Tab	智能缩进或者自动完成（即输入前缀后会列出相关的关键字和属性）
CTRL+Space建议改成ALT+\	(show completions)显示自动完成
CTRL + \	(show calltip)显示函数的参数使用方式
Ctrl+]	缩进代码块(Indent Region)
Ctrl+[	取消代码块缩进(Dedent Region)
Alt+3	注释代码块(Comment Region)
Alt+4	取消代码块注释(Uncomment Region)



Format子菜单

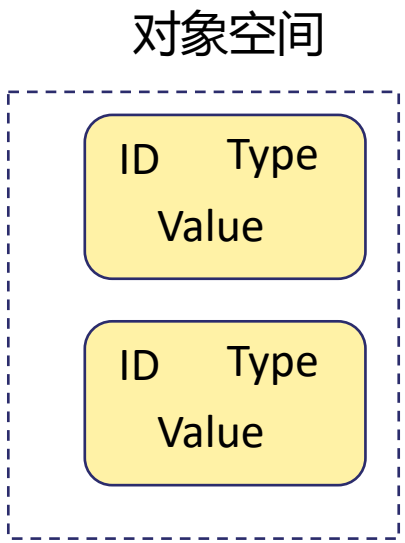
- **Indent/Dedent Region: 缩进控制**
- **Comment/Uncomment Region: 注释控制**
- Strip Trailing Whitespace: 每行最后多余的空格移走

主要内容

- 编程（补充的内容）和python语言概述
- **对象和赋值语句**
- 数字类型和算术运算符
- 字符串和相关运算符

数据类型和对象

- 数据(值， value）被保存在某种存储（内存或外存）中
  - 以二进制方式保存，基本存储单元为字节
  - 104 → 01101000
  - 'h' → 01101000
- 类型：比如整数、实数和字符串
  - 描述了一组相关值（值的取值范围），以及可以对这些值执行的一组操作
  - 有些程序语言要求程序员在代码中显式地声明数据的类型
  - python语言不要求声明数据类型
- 对象(object)：Python中各种数据的抽象
- 对象有**三个基本的属性**，除此之外，对象还可包括其他属性
  - 标识ID(identity): 对象一旦创建，在其生命期间其**ID不再改变**，可以看成该对象在内存中的地址 (cython实现)
  - 类型(type)：决定了对象可能取值的范围及支持的操作。对象的**类型不可变**，即Python是一种强类型语言
    - 对象是类型的一个实例（instance）
  - 值(value): **值可以被改变或不可变**，决定于其类型
    - 不可变(immutable)：有些对象一旦创建其值不可变，比如数字类型、字符串、元组等
    - 可变(mutable): 对象的值可以改变，比如列表、字典等



基本数据类型

- Literal(字面值/**字面量**)：表示某个内置对象类型的值，在词法和语法分析时识别其类型
- **表达式**(expression)：一个值，或者多个值通过多个运算符(operator)运算之后的结果

  (3 * 4 + 5) * 6
- NoneType类型仅仅有一个值，即None，常用于函数返回值

	类型	描述	示例
数字类型	int	整数	42， 20493， 0
	float	实数(浮点数)	7.35, 4., 6.022e23
	complex	复数	1+2j, 3.4-1j
	bool	布尔	True, False
	NoneType	空类型	None
	str	字符串	“Hello.”, ‘abc 1 2‘, ''
	list	列表	[1, 2, 3]
	tuple	元组	(1, 2, 3)
	dict	字典	{'name':'tony', 'age':22}
	set	集合	{1, 2, 3}

变量和赋值语句

python注释： #字符后的内容不会被解释器解释

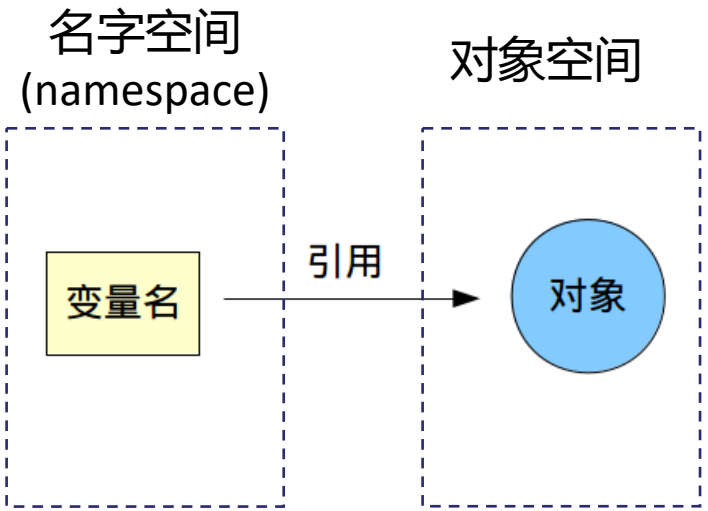
```
x = 3
a = 3 + 4
a = a + 3
b = a
b = 3.14
c = d + 8  # NameError
```

- 对象：Python中各种数据的抽象
- 表达式(expression): 各个对象通过运算符(operator)运算之后的结果 (3 * 4 + 5) * 6
- **变量(variable)**: 表示对于某个对象的引用（reference）
  - 变量有一个名字，称为变量名
  - 通过**赋值语句(assignment statement)**给变量赋值，类似于给对象取一个别名(alias)
    - variable = expression（LHS = RHS，left-hand side/right-hand side)
    - 首先计算RHS的表达式的值（对象），然后LHS的变量指向该对象（即值赋值给变量）
    - 变量出现在LHS(Left Hand Side),给该变量赋值，即保存RHS所对应对象的引用（地址）
    - 变量出现在RHS(Right Hand Side) 处时表示引用该变量所指向对象的值
    - 变量在使用前（出现在表达式）必须有定义（赋值）
  - 变量在赋值之前**无需声明其类型**，变量的类型为所指向对象的类型
- Python是一种**动态类型**语言
  - 变量的类型是可以随时变化的
  - 对象的类型本身是不会变的

Namespaces are one honking great idea -- let's do more of those!
The Zen of Python（import this)

- 名字空间通过字典dict实现，字典保存的是key:value对，

  key为变量的名字，而value为对应的对象



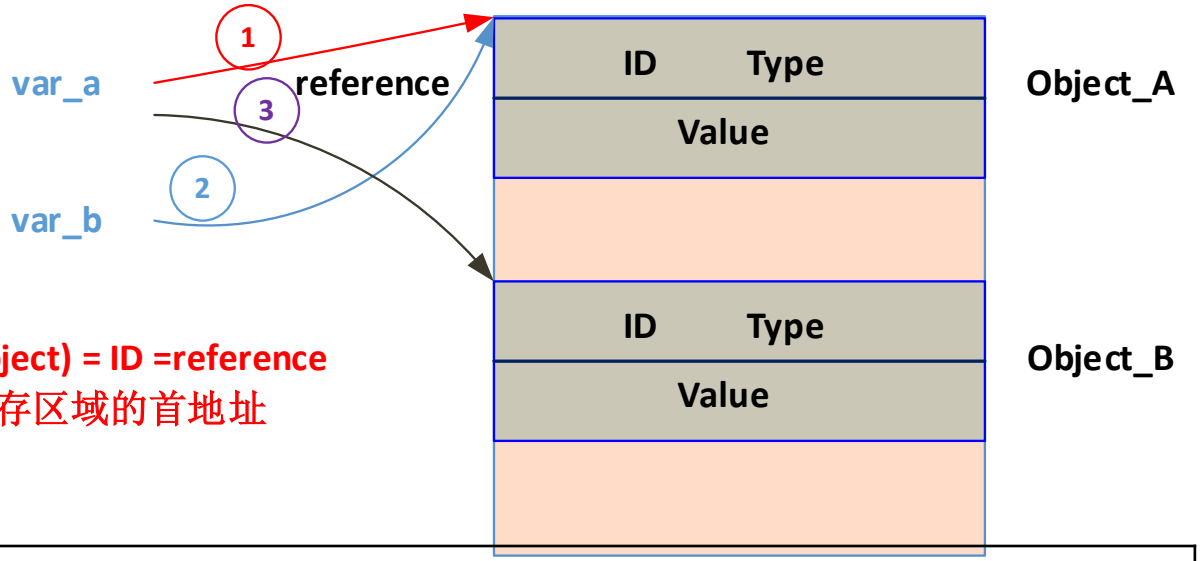
变量和赋值语句

- 变量表示某个对象的引用，允许多个变量指向同一个对象
- a = a + 3， 变量a指向了原来变量a的值与3相加后的另一个对象
- b = a 相当于给a所指向的对象取了另外一个**别名b**

```
a = 3 + 4
a = a + 3
b = a
b = 3.14
```

```
>>> a = 3 + 4
>>> id(a)
1688352877040
>>> type(a)
<class 'int'>
>>> isinstance(a, int)
True
>>> a == 7
True
>>> a = a + 3
>>> id(a)
1688352877136
>>> b = a
>>> a is b
True
```

var_a = Object_A # 1
var_b = var_a # 2
var_a = Object_B # 3



CPython实现中 id(object) = ID =reference
=存放object的内存区域的首地址

id(object)	返回object的ID
type(object)	返回object的类型
isinstance(obj, cls)	测试对象obj是否为**指定类型cls或其子类型**的实例，返回True或False, cls可为int/float/str等，它指向某个类对象。注意与type(obj) == cls的区别
obj1 == obj2	判断obj1和obj2的值是否相等，结果为True或False
obj1 is obj2	判断obj1和obj2是否同一对象, 结果为True或False。id(obj1) == id(obj2)

复合赋值与链式赋值

- 复合赋值（augmented assignment):

  var op= expression， op为算术运算符(+ - * / // % **)或位运算符(& | ^)

  - 注意运算符和=之间没有空格，相当于 **var = var op expression**

  total += item 等价于 total = total + item

  total + = item # 语法错误SyntaxError: invalid syntax

  total *= item 等价于 total = total * item

  total **= item 等价于 total = total ** item

  - 简化代码，精炼程序
  - 对于可变类型的变量var而言，复合赋值为IN-PLACE计算，提高效率

- 链式赋值（chained assignment)： **var1 = var2 = … = expression**

  - 右结合，即首先 var2 = …. ; var1 = var2

  x = y = 123 等价于 y = 123; x = y

✓Python一般一行一个语句，但也可在同一行中放置多条语句，语句之间使用分号“;”分割，但为可读起见，不建议

变量名

- 变量的名字必须符合标识符(identifier)的语法
- 标识符必须以字母或下划线开头，其后的字符可以是字母、下划线或数字
  - 变量名中不能有空格及标点符号（括号、引号、逗号、斜线、反斜线、冒号、句号、问号等）
  - 非法的标识符： 99var It'OK first-class
- 标识符长度至少为1
- 标识符对英文字母的大小写敏感，例如student和Student是不同的变量
- 一些特殊的标识符保留为Python关键字(keyword)，不能用作变量名

```
>>> help('keywords')

Here is a list of the Python keywords.  Enter any keyword to get more help.

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

变量名

- 不建议使用系统内置的模块名、类型名或函数名以及已导入的模块名及其成员名作变量名，否则原来的对象就无法通过该名字访问了
- 建议使用有意义的名字，比如area、keys等。单字母变量名(如i, j, k)仅用在较短的循环等结构
  - 如果变量名与内置的对象名字重复，可在最后添加下划线，比如sum_
- 建议使用小写字母表示变量名，多个单词时以下划线隔开：lower_case_with_underscores
- 避免使用小写字母l和大写字母O
- 以**下划线开头的变量**一般表示在某个范围中内部（临时）使用的变量。**python标准控制台中使用变量_，记录最近的表达式计算的结果**

```
>>> print = 10  # 内置函数print
>>> print
10
>>> print("Hello Word!")
Traceback (most recent call last):
  File "<pyshell#62>", line 1, in <module>
    print("Hello Word!")
TypeError: 'int' object is not callable
>>> del print
>>> print("Hello Word!")
Hello Word!
```

del语句： del var

del var1, var2

- 将变量var与原有对象解绑，从相应的名字空间中删除
- 再访问var时会抛出NameError异常
- 如果很想用print这个名字，可将其稍作改变为print_

```
>>> x = 4
>>> x
4
>>> del x
>>> x
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    x
NameError: name 'x' is not defined
```

主要内容

- 编程（补充的内容）和python语言概述
- 对象和赋值语句
- **数字类型和算术运算符**
- 字符串和相关运算符

整数(int)

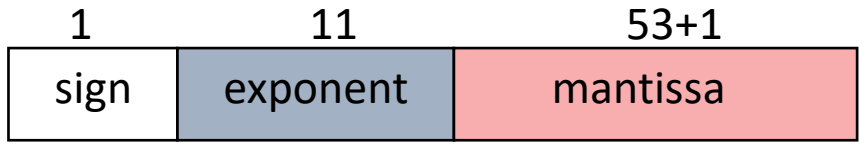
- 整数字面量
  - 十进制整数: 0、-1、9、123
    - 整数0前面可填充0(如000)，但**非0整数不允许前面填充0**
  - 十六进制(hex)整数: 0x或0X开头，后面为十六进制数字，即为0-9以及[a-f]（abcdef，也包括其大写形式ABCDEF）如0x10、0xfa、0xabcdef
  - 八进制(octet)整数:以0o开始后面为八进制数字即0-8，比如0o35、0o11
  - 二进制(binary)整数:以0b开头如，0b101、0b100 、 0b10010111
- 整数类型int可以表示**任意大的数值，只要内存允许**
- **拓展：** python3.6之后，支持在数字中间位置使用单个下划线作为分隔提高数字的可读性，类似于数学上使用逗号作为千位分隔符
  - 单个下划线可以出现在中间任意位置，但不能出现在开头和结尾位置，也不能使用多个连续的下划线

```
>>> a = 999999999999999999999999999999
>>> print(a * a * a)
999999999999999999999999999997000000000000000000000000000002999
999999999999999999999999999
```

```
>>> -1
-1
>>> 007
SyntaxError: invalid token
>>> 000
0
>>> 0x10
16
>>> -0x10
-16
>>> 0xfa
250
>>> 0XABCDEF
11259375
>>> 0o35
29
>>> 0o11
9
>>> 0b100
4
>>> 4_758_000
4758000
```

（有限精度）浮点数（float）

- 内置类型浮点数(float)为有理数的一个子集，任意实数可近似为某个浮点数
- 浮点数字面量：
  - 小数表示：3.14　10.　.001
  - **科学计数法**：15e-2 ($=15*10^{-2}=0.15$)　3.14e-10　1.0E100
- **拓展：**与其他语言一样，浮点数在计算机内部采用二进制的科学计数法表示（浮点数名称的由来），即IEEE 574双精度(64bit double)
  - 尾数(mantissa)：归一化, 二进制的1.xxxx，最高位的1可以省略
  - 指数(exponent)，可以为正数和负数



$$sign * mantissa * 2^{expoent}$$

- 由于浮点数内部表示的位数限制，在转换和计算时会有精度误差
  - 十进制的有限小数转变为二进制时可能是无限循环小数！
    比如：$0.1_{(10)}=0.0001100110011001\ldots_{(2)}$

```
>>> import sys
>>> sys.float_info
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308,
min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15,
mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

```
>>> 3.14
3.14
>>> 10.
10.0
>>> .45
0.45
>>> 0.001
0.001
>>> 15e-2
0.15
>>> 31.4e-11
3.14e-10
>>> 1.0E100
1e+100
>>> 0.1
0.1
>>> 0.1+0.1
0.2
>>> 0.1+0.1+0.1
0.30000000000000004
```

布尔(bool)、复数(complex)和NoneType

```
>>> issubclass(bool, int)
True
>>> isinstance(True, bool)
True
>>> isinstance(True, int)
True
```

- 布尔(Boolean)类型bool：
  - 只有两个值，分别为True和False
  - bool可以看成整数类型的特例，True和False分别对应整数1和0
- 复数(complex)类型：
  - 一个复数包括实部（real）和虚部（imaginary）两个部分
  - **虚数字面量**：浮点数或者整数，后面为j（或者J），**中间不能有空格**

```
3 + 4j   1.1 + 2.2j    3 + 4.2j   3-4.2j   4j    0j
```

    - 注意
      - 1j和j的区别：1j表示虚部为1，而j表示名字为j的变量
      - 4*j表示4和变量j相乘
- NoneType类型：
  - NoneType类型只有一个值，即None

运算符

- 运算符：一种特殊符号（比如+），表示要对一个或者多个值（称为运算数）执行的运算
  - 只有1个运算数的运算符称为一元运算符
  - 有2个运算数的运算符称为二元运算符
  - 有3个运算数的运算符称为三元运算符
- 算术运算符：进行算术运算的运算符

- python不支持其他语言(C/Java)等支持的**++(加1)和--(减1)**运算

运算符	含义	示例
+, -	一元(只有一个运算数）的符号运算	+8, -8
+, -	加法和减法	35 + 4 , 2.5 + 3.5
*	乘法	(35 + 4) * 5, 3.4 * 5.2 + 4
**	幂运算	4 ** 2 = 16 4 ** 0.5 = 2.0
/	真除法或浮点除法, 结果为浮点数	5 / 2 = 2.5 4 / 2 = 2.0 10 / 3.2 = 3.125
//	整除, 求整商, 结果可为整数或浮点数 • 两个整数整除, 结果为整数 • 整除的数有浮点数时, 结果为浮点数	5 // 2 = 2 4 // 2 = 2 5.1 // 2.2 = 2.0
%	取余或求模, 有浮点数时结果为浮点数 a % b = a – a // b * b	5 % 2 = 1 4 % 2 = 0 5.1 % 2.2 = 0.699999999999993

```
>>> --------1
1
>>> --1
1
>>> +--+--1
-1
```

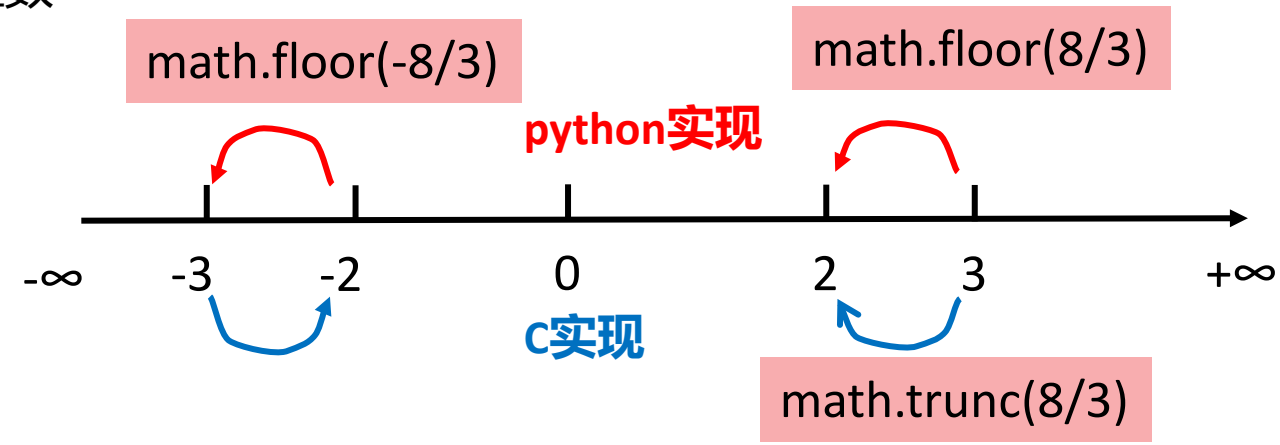
除法类运算符

浮点除法、整除和求模运算：/ // %

- 除数不能为0，抛出异常**ZeroDivisionError。**比如 2 / 0　2 // 0　2 % 0
- 浮点除法也称“真除法”，**结果为浮点数**　5 / 2 = 2.5　　4 / 2 = 2.0
- 整除 // ：求整商 。有浮点数时结果为浮点数，否则整数

  8 // 3 = 2　8 // 3.0 = 2.0

- 取余或求模 %，有浮点数时结果为浮点数，否则整数
  - a % b == a – a // b * b
  - 8 % 3 = 2　　8 % 3.0 = 2.0　8 % 3.5 = 1.0



- 求模和整除运算符的应用
  - 获取个位数　　230857 % 10 = 7
  - 获取百位以上数字　230857 // 100 = 2308
  - 判断奇偶　　230857 % 2 == 0　False
  - 解决周期问题　　start in [0, 6]，n天之后星期几？　(start + n) % 7
- **拓展：如果整除和求模运算时，其结果为负数时如何计算？负数的情形比较少见**
  - Python的实现为向负无穷方向舍入（floor()函数）
  - C/C++、Java为向原点(0)方向舍入（trunc()函数）
  - 对于结果为正数而言，两种实现的结果一致

```
>>> 8 // 3
2
>>> 8 % 3
2
>>> -8 // 3
-3
>>> -8 % 3
1
```

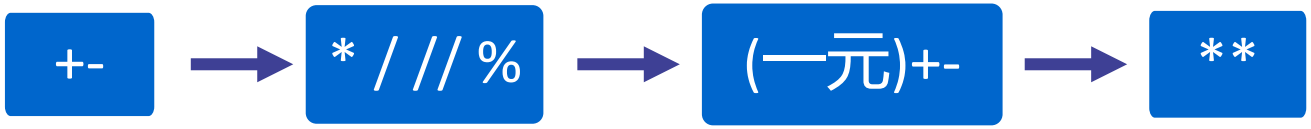
算术运算符： 优先级及混合运算

- 算术运算符支持两个不同的**数字类型**的对象之间的运算，即混合运算，比如浮点数+整数
  - python会自动类型提升：将低级别的运算数提升为更高级别的类型后再进行运算，但注意仅仅针对数字类型的对象之间的算术等运算
  - 如果其中包含complex对象，则其他数字类型转换为complex对象，结果为complex对象
  - 如果其中包含float对象，则其他数字类型转换为float对象，结果为float对象

类型提升：bool → int → float → complex

```
>>> True + 4 + 4.5
9.5
>>> 2 ** 3 ** 2
512
>>> (2 ** 3) ** 2
64

>>> [2 * 3] * 2
[6, 6]
```

- 有多个运算符时如何决定运算的顺序?
  - 算术运算符具有不同的优先级顺序，优先级高的优先计算
  - 对于同级的运算符，除了幂运算**为右结合运算外，其余同级为左结合运算
  - 可通过**小括号改变运算顺序**，小括号可嵌套
    - 注意：中括号[ ]和大括号{}不用于数学计算，而是用于定义列表、字典以及集合等容器对象



四组运算符优先级顺序：从低到高

对象的属性和方法

- 对象obj有3个基本属性ID、类型和值:
  - id(obj); type(obj); obj
- 对象还可包括其他属性
  - 对象的属性(attribute):
    - 通过obj.attr的形式访问, ob为指定的对象，而attr为对象的属性名
    - 保存了该对象的指定属性
  - 对象的方法（method）
    - 一种**特殊类型的属性**，其保存的是一个可调用的对象，与函数类似，**可以调用该方法**
    - 通过obj.method访问, method为对象的方法名
  - 一个对象的属性和方法的名字属于该对象内部的名字空间
- (qualified)**限定名**: 通过dot路径限定其名字所在空间
- **未限定名**：前面没有dot路径，表示当前名字空间
- c.real中c为未限定名，而c.real中的real为限定名

```
>>> c = 8 + 10j
>>> real = 7
>>> real
7
>>> c.real # 查看复数实部
8.0
>>> c.imag # 查看复数虚部
10.0
>>> c.conjugate() # 返回共轭复数
(8-10j)
>>> type(c)
<class 'complex'>
>>> type(c.real)
<class 'float'>
>>> type(c.conjugate)
<class 'builtin_function_or_method'>
```

主要内容

- 编程（补充的内容）和python语言概述
- 对象和赋值语句
- 数字类型和算术运算符
- **字符串和相关运算符**

字符串(str)

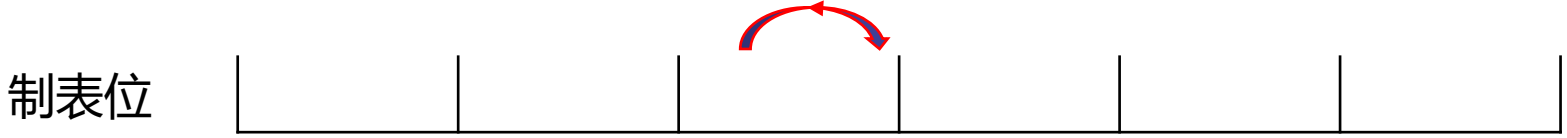
- 字符串表示一串字符组成的文本，通过**字符串界定符**定义，以某个字符串界定符开始，以同样的字符串界定符结束
- 其他语言一般用双引号定义字符串字面量，Python的字符串界定符可以使用**单引号**和**双引号**
  - 一般建议采用单引号定义
  - 如果字符串中有双引号，则可使用单引号定义，反之亦然

  'abc'　　"Python程序设计基础"　　"What's your name?"　　'"Thank you!" she said. '
- 两个以上连续（中间可用空格等分隔）的字符串字面量被合并为一个字符串　'12' '34' 相当于'1234'
- 内置函数**len(s)**：s为字符串时，返回字符串中的字符个数
- 字符串可以为**空字符串**，即不包含任何字符，可用两个连续的界定符定义，比如 ''
- 字符串字面量定义的限制：
  - 短字符串定义：单引号和双引号定义的字符串不能跨越多行
  - 字符串中的文本如果出现了字符串界定符本身怎么办？ 见下一页slide

编程经常出现的问题是采用了全角字符，包括中文标点符号

```
# 下列字符串定义会报错 SynatxError: EOL while scanning string literal
s1 = 'a line with no matching delimiter
s2 = ' be careful of Chinese punctuation ’
```

字符串转义



- 如果一个字符串内容中出现界定符(如单双引号)本身，则需要通过在引号前加上**转义字符**\表示其不是字符串界定符，而是普通的引号
  - \' 表示单引号本身， \"表示双引号本身
  - \\表示单个的反斜杠
- 还可通过转义序列描述一些控制字符

转义序列	含义	转义序列	含义
\'	单引号	\"	双引号
\\	反斜杠	\t	制表符Tab
\n	换行符 Newline	\r	回车CarriageReturn
\a	响铃BEL	\b	后退Backspace
\f	FormFeed，输出到打印机时换页	\v	垂直制表VT

```
>>> print('\'hello\' and "hello"')
'hello' and "hello"
>>>print('name\tphone\taddr\nidle\t123344\tidle\'house')
name    phone   addr
idle    123344  idle'house
```

```
>>> print("\\hello\nhow\tare \"you\"?\\\\")
\hello
how     are "you"?\\
```

字符串转义

- \表示**后面的字符可能不是原有的含义**，需要特别对待，即可能具有特殊含义
  - 如果后面的字符的确有特殊的含义，即是合法的转义序列，则转义字符移走，使用特殊的含义
    - \后面是单引号、双引号和反斜杠时，表示后面的字符不用于字符串界定和转义，而是字符本身
    - \后面是n/t/r/a/b/v/f时，表示对应的控制字符（换行、制表等）
    - \后面跟着数字或x/u/U等表示根据码点描述的相应字符（后面会展开）
  - 如果后面的字符并没有特殊的含义，则**转义字符仍然保留**，比如 '\c'包含两个字符：反斜杠和c

```
>>> print('d:\dlmao\python')
d:\dlmao\python
>>> print('d:\tony\network')
d:      ony
etwork
>>> print('d:\\dlmao\\python')
d:\dlmao\python
>>> print('d:\\tony\\network')
d:\tony\network
```

长字符串

- 短字符串定义：单引号和双引号定义的字符串不能跨越多行
  - 通过\n定义多行字符串
- 长字符串：
  - 采用**三(单/双)引号**'''或"""作为字符串界定符时
  - 长字符串可以跨越多行
  - 除了可跨越多行外，其他与短字符串完全一样，也支持字符转义

- python语言支持一般采用#进行单行注释，#之后同一行的内容忽略掉
- python语言本身不像C等支持多行注释，但是由于表达式执行后并没有任何side effect，可以采用长字符串来支持**多行注释**

```
>>> print('''first line
line 2
line 3
last line.''')
first line
line 2
line 3
last line.
```

```
>>> print('first line \nline 2\nline 3\nlast line.')
first line
line 2
line 3
last line.
>>> print('''first line \nline 2
line 3
last line.''')
first line
line 2
line 3
last line.
```

原始字符串

- 短字符串和长字符串的内容中如果有许多字符需要进行转义会比较繁琐

```
>>> print('''要让变量a指向字符串"C:\\Program Files (x86)\\Python35\\bin"
应该这样定义：a = "C:\\\\Program Files (x86)\\\\Python35\\\\bin" ''')
要让变量a指向字符串"C:\Program Files (x86)\Python35\bin"
应该这样定义：a = "C:\\Program Files (x86)\\Python35\\bin"
```

- 原始(raw)字符串定义：
  - 字符串界定符前面**加字母r或R**，界定符之间的所有内容作为原始字符串的内容
  - 原始字符串定义中，特殊字符不需要转义

```
>>> print(r'''要让变量a指向字符串"C:\Program Files (x86)\Python35\bin"
应该这样定义：a = "C:\\Program Files (x86)\\Python35\\bin" ''')
要让变量a指向字符串"C:\Program Files (x86)\Python35\bin"
应该这样定义：a = "C:\\Program Files (x86)\\Python35\\bin"
```

原始字符串的局限性

拓展的内容

- 拓展：python解释器在进行词法分析时会**解释转义字符**以确定原始字符串是否结束，但是一旦**确定了原始字符串的开始和结束位置**后，之间的所有内容都作为原始字符串的内容，即**转义符又失去了原来的意义**
- 不是所有的字符串都可以采用原始字符串定义，比如最后一个字符为\时

```
>>> print(r'c:\')

SyntaxError: EOL while scanning string literal
>>> print(r'c:\\')
c:\\
>>> print(r''hello'')
SyntaxError: invalid syntax
>>> print(r'\'hello\'')
\'hello\'
>>> print('\'hello\'')
'hello'
```

字符串对象支持的算术运算符： + * %

- **加法运算符：s1 + s2**
  - 字符串与字符串相加，生成一个新的字符串，由两个字符串合并而成（**concatenate**）
  - 字符串不能与其他类型对象相加
- 乘法运算符：s * n 或 n * s
  - 字符串与整数相乘，生成一个新的字符串，相当于字符串的内容重复多次。整数小于等于0时生成空字符串''

```
>>> a = 'abc' + '123'          #生成新对象   'abc123'
>>> print(a)
abc123
>>> 'abc' + 4
Traceback (most recent call last):
  File "<pyshell#261>", line 1, in <module>
    'abc' + 4
TypeError: Can't convert 'int' object to str implicitly
```

```
>>> print('abc' * 4)
abcabcabcabc
>>> print('-' * 25)
-------------------------
>>> print(4 * '*-')
*-*-*-*-
```

字符串对象支持的算术运算符：%

format_string % value

format_string % (value1, value2, …)

一个或多个对象按format_string给出的格式要求转换为字符串。产生类似C语言的printf格式化输出

- 格式化字符串是格式化的模板，由固定文本以及格式说明符混合而成
  - 格式说明符： **%[flags][width][.precision]type**
    - 说明后面的参数如何转换为字符串
  - 如果有多个格式说明符，则采用第2种格式，%后面的对象是元组（tuple）对象，它是由多个值组成的容器对象，在格式化时按照位置顺序与前面的格式说明符对应
- 最后格式化后的字符串中：
  - 固定文本原封不动
  - 格式说明符部分被一个字符串替换，该字符串由运算符%后面的对应位置参数(值）根据相应的格式说明转换而来

```
>>>'Hi %s' % 'idle'
'Hi idle'
>>> 'x=%d, y=%5.2f' % (2, 2**3)
'x=2, y= 8.00'
```

格式说明符

格式说明符： % [flags] [width] [.precision] type

- 格式字符type：要求与待格式化的值类型匹配

```
>>>'Hi %s' % 'idle'
'Hi idle'
>>> 'x=%d, y=%5.2f' % (2, 2**3)
'x=2, y= 8.00'
>>> '%d%%' % 30.5
'30%'
```

格式字符	说明	格式字符	说明
%s	任意对象obj，输出str(obj)	%x或%X	整数，输出十六进制字符串，X表示大写
%r	与%s类似，只是为repr(obj)	%e	数字，输出科学计数法表示，基底为e
%c	整数n或字符。如果为整数，首先调用chr(n)得到UNICODE字符	%E	与%e类似，只是基底为E
%d	数字，首先转换为整数，再转换为十进制字符串	%f、%F	数字，输出小数点表示法, 精度缺省为6
%i	与%d类似，输出十进制整数字符串	%g或%G	科学计数法(e/E)或浮点数(根据显示长度)
%o	整数，输出八进制字符串	%%	字符%

- %s %r对应的值可是任意类型
- %c 对应的值为整数或字符，%o %x对应值为整数
- 其他格式字符要求对应的值为数字类型

```
>>> "%d" % "555"
Traceback (most recent call last):
TypeError: %d format: a number is required, not str
```

格式说明符

格式说明符： % [flags] [width] [.precision] type

- 宽度和精度可选
  - 宽度指最小宽度。如果实际的值格式化后的宽度不够时填充。但是如果值实际宽度超过了最小宽度时，也不会截取，而是允许超过最小宽度
  - 精度：一般用于浮点数，表示小数点后的位数，不指定精度时缺省为6位
- flags: 可选，表示对齐和填充等,格式为 [-][0][+][#]
  - 默认右对齐。如果为-表示左对齐
    - 只有指定了最小宽度，且实际的值格式化后的宽度不够时，对齐和填充才有意义
    - 填充：默认填充空格，如果为0表示填充0
  - 符号：如果为+表示对于正数也要添加符号
  - #：type为xXo等(转换为十六进和八进制)时前面加上相应的进制表示前缀(0x或0o等)

```
>>> '%f' % 123.4
'123.400000'
>>> '%.2f' % 123.4
'123.40'
>>> '%4.2f' % 123.4
'123.40'
>>> '%8.2f' % 123.4
'  123.40'
>>> '%-8.2f' % 123.4
'123.40  '
```

```
>>> '%08.2f' % 123.4
'00123.40'
>>> '%+8.2f' % 123.4
' +123.40'
>>> '%s' % 123.4
'123.4'
>>> '%8s' % 123.4
'   123.4'
>>> '%-8s' % 123.4
'123.4   '
```

```
>>> '%d' % 123
'123'
>>> '%8d' % 123
'     123'
>>> '%-8d' % 123
'123     '
>>> '%08d' % 123
'00000123'
```

```
>>> '%o' % 123
'173'
>>> '%#o' % 123
'0o173'
>>> '%x' % 123
'7b'
>>> '%#x' % 123
'0x7b'
>>> '%#X' % 123
'0X7B'
```

字符串格式化

- 标题行和数据行可以指定相同的宽度和对齐方式

```
print('%-15s %-30s %-15s %s' % ('ID', 'Name', 'Score', 'Memo'))
print('%-15s %-30s %-15.2f %s' % ('00001506', 'Guido van Rossum', 100, 'Founder of Python'))
```

```
ID              Name                           Score           Memo
00001506        Guido van Rossum               100.00          Founder of Python
```

- format_str % value, **如果value包含了其他运算符时需要注意运算的优先级顺序**
  - * % / // 具有相同的优先级，优先级相同时采用从左到右的顺序计算

```
>>> '%d' % 12 * 4
'12121212'
>>> '%d' % 12 + 4
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    '%d'%12 + 4
TypeError: can only concatenate str (not "int") to str
>>> '%d' % (12 * 4)
'48'
```

%和*的优先级一样，采用左结合

返回某种类型的实例对象：构造函数法

- int/float/bool/complex/str等内置**类对象**，分别对应整数、浮点数等类型
  - 这些类对象**可以像函数一样调用**，称为构造函数，返回一个该类型的实例对象

int() **返回整数0**	**int(x) x为数字类型(不包括复数)** **True/False转换为1/0** **浮点数取(原点截取)整数部分**	**int(x, base=10) x为字符串** **缺省x为十进制整数字符串(字节串bytes/bytearray)**
float() 返回浮点数0.0	float(x) x为数字类型(不包括复数) True/False转换为1/0	float(x) x为字符串 仅支持十进制，将**十进制的整数字符串或浮点数字符串转换**为浮点数
bool() 返回False	bool(x) 非0数字类型、非空序列类型转换为True，否则转换为False	
str()返回空字符串	str(obj) 将对象转换为字符串	

```
int(), int(3.14), int('314'), int('ff',16)                    # (0, 3, 314, 255)
float(), float(3), float('-3'), float('3.14'), float('10e2')  # (0.0, 3.0, -3.0, 3.14, 1000.0)
bool(), bool(3), bool(0), bool(3.14)                          # (False, True, False, True)
str(), str(3.14), 'abc' + str(4)                              # ('', '3.14', 'abc4')
```

内置函数str与repr

str(obj)	返回一个对人友好的printable字符串，让用户了解该对象的相关信息。print函数的参数可为任意对象obj，相当于print(str(obj))
repr(obj)	返回描述对象的内部表示(representative)的字符串，让程序员了解该对象。该字符串存储的一般是可以写在程序中的字面量。制表、回车、换行字符以\t\r\n表示，其他ASCII控制字符以\x...形式描述

对象(obj)	print(str(obj))	print(repr(obj))
3	3	3
True	True	True
[1, 2]	[1, 2]	[1, 2]
'123'	123	'123'
r'1\2\\3'	1\2\\3	'1\\2\\\\3'

```
>>> text= r'c:\tmp\python 程序设计'
>>> print(text)
c:\tmp\python 程序设计
>>> text
'c:\\tmp\\python 程序设计'
>>> print(repr(text))
'c:\\tmp\\python 程序设计'
```

- 交互式console：如果执行的语句为表达式obj，且其值**不为None**时调用**print(repr(obj))**。
  对于字符串而言，输出结果为采用转义方式描述的字符串字面量

内置函数input

内置函数：input(prompt=None)

- 可不传递参数；也可传递一个参数，此时首先输出prompt
- 等待用户输入，直到用户按**回车健**结束；如果在输入回车之前，按<Ctrl-C>，则抛出异常KeyboardInterrupt；如果按<Ctrl-D>则抛出异常EOFError
- 返回用户输入的字符串（不包括最后的回车）

```
>>> x = input()
45.7
>>> x
'45.7'
>>> x = input('Please input:')
Please input:3
>>> print(type(x))
<class 'str'>
>>> x
'3'
>>> int(x)
3
>>> x = int(input('Please input:'))
```

问题：用户输入一个三位自然数，计算并输出其百位、十位和个位上的数字。

```
x = int(input('请输入一个三位数: '))
a = x // 100
b = x // 10 % 10
c = x % 10
print(a, b, c)
```

查看帮助

- 在交互式控制台，可以使用**help和dir函数**查看帮助信息

dir()	列出当前名字空间中的所有名字
dir(object)	列出object相关的属性和方法列表，其中下划线(_)开头的名字表示内部使用，如dir(str), dir('hello')
help(object)	查看对象object相关的帮助，比如help(print)查看函数print的相关帮助，help(3.14)查看浮点数支持的属性和方法, help(str)查看类型为str的字符串对象的属性和方法
help()	进入交互式的Help系统，键入quit退出帮助系统
help('topic')	查看交互式help系统的某个主题(topic) 相关的帮助，比如help('if')查看if语句语法

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'y']
>>> dir(str)
[..., 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>> help('if')
...
   if_stmt ::= "if" expression ":" suite
               ( "elif" expression ":" suite )*
               ["else" ":" suite]
```