PYTHON程序设计

计算机学院 王纯

基础语法

基础语法

- 语法的表示
- 标识符、变量及其赋值
- 基本数据类型
- 运算符与表达式
- 基本输入输出

语法的表示

- ✓ BNF范式（Backus-Naur Form） 作为描述计算机语言的数学方法由 John Backus（Fortran语言之父）提出，Peter Naur将之改进并在 Algol 60 中用于描述语法。
- ✓ BNF 被用来形式化定义程序设计语言的语法，以使其规则没有歧义。 广泛的应用在编程语言、指令集、通信协议甚至自然语言（其中的一部分）的语法描述上。
- ✓ 由一组可推导的产生式组成，如下形式：

symbol : : = alternative1 |
alternative2 . . .

- 自然语言
  - 字 ->单词 ->句子 ->语言
- 计算机语言
  - 字符 ->词法单位(token) ->程序 ->语言
- BNF（Backus Naur Form）
  - 一种形式化描述语法的工具， 一种表示方法

G:＜英语句子＞ ::= ＜主语＞＜谓语＞＜宾语＞
＜主 语＞ ::= ＜冠词＞＜形容词＞＜名词＞
＜冠 词＞ ::= the
＜形容词＞ ::= big
＜谓 语＞ ::= ＜动词＞
＜动 词＞ ::= ate
＜宾 语＞ ::= ＜冠词＞＜名词＞
＜名 词＞ ::= peanut
＜名 词＞ ::= elephant
文法的组成
文法开始符 规则 / 产生式
非终极符 终极符

语法的表示

例1 标识符

- 〈标识符〉 ::= 〈字母〉
- 〈标识符〉 ::= 〈标识符〉 〈字母〉
- 〈标识符〉 ::= 〈标识符〉 〈数字〉
- 〈字母〉 ::= _|a | b | … | z|A|…|Z
- 〈数字〉 ::= 0 | 1 | 2… | 9

例3 算术表达式

- <expression>::= <term> | <term> "+" <expression>

<term>::= <factor> | <factor> "*" <term>

<factor>::= <constant> | <variable> | "(" <expression> ")"

<variable>::= "x" | "y" | "z"

<constant>::= <digit> | <digit> <constant>

<digit>::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

例2—无符号整数集合的文法

G: N ::= D N ::= ND

D ::= 0 D ::= 5

D ::= 1 D ::= 6

D ::= 2 D ::= 7

D ::= 3 D ::= 8

D ::= 4 D ::= 9

符号“|”表示“或者”文法可简写成:

G: N ::= D| ND

D ::= 0| 1| 2| 3| 4| 5| 6| 7| 8| 9

Python的语法定义基于EBNF和PEG：10. Full Grammar specification — Python 3.10.6 documentation

标识符、变量及其赋值

标识符

由程序员定义的名字；
允许采用大写字母、小写字母、数字、下划线(_)和汉字等字符，但标识符的首字符不能是数字，中间不能出现空格；
大小写敏感；
不能与保留字相同；
允许使用汉字标识符，但我们不建议这么做。

Python中以**下划线开头**的标识符有特殊意义， 一般应避免使用相似的标识符

（1） 以单下划线开头的标识符（_width） 表示不能直接访问的类属性。 另外，也不能通过 from xxx import * 导入

（2） 以双下划线开头的标识符（如__add）表示类的私有成员

（3） 以双下划线开头和结尾的是Python里专用的标识， 例如 __init__() ‘ 表示构造函数

合法的标识符：Python_3_7, python_你好, _python_ABC
五个不同的标识符：Python_3、python_3、PYTHON_3、PyThOn_3、pYtHoN_3

标识符、变量及其赋值

保留字

编程语言内部定义并保留使用的标识符。可以在命令行查看

```
>>> import keyword          #导入keyword模块
>>> keyword.kwlist          #调用kwlist显示保留关键字列表
```

False	None	True	and	as	assert	async
await	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

标识符、变量及其赋值

变量 表示（或指向）特定值的标识符；

不需要事先声明(这一点和C语言很不一样)，变量的赋值操作即是变量的声明和定义的过程；

x = 5

常量 在程序执行过程中不能改变的数据，比如：

5，“abc”

没有命名常量，所以不能像C语言那样给常量起一个名字。

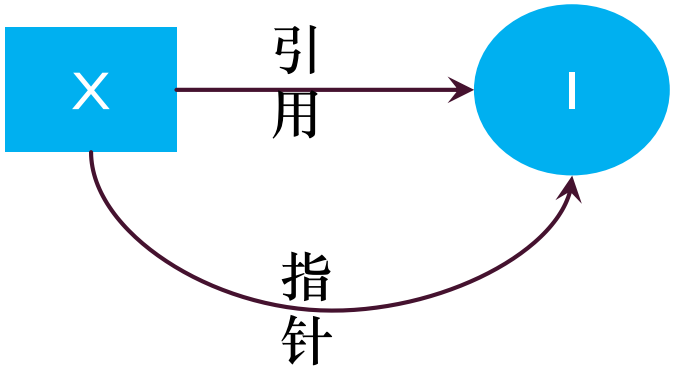
1） 变量名必须是一个有效的标识符
2） 变量名不能使用Python中的保留字
3） 慎用小写字母**l** 和大写字**O**
4） 应选择有意义的单词作为变量名

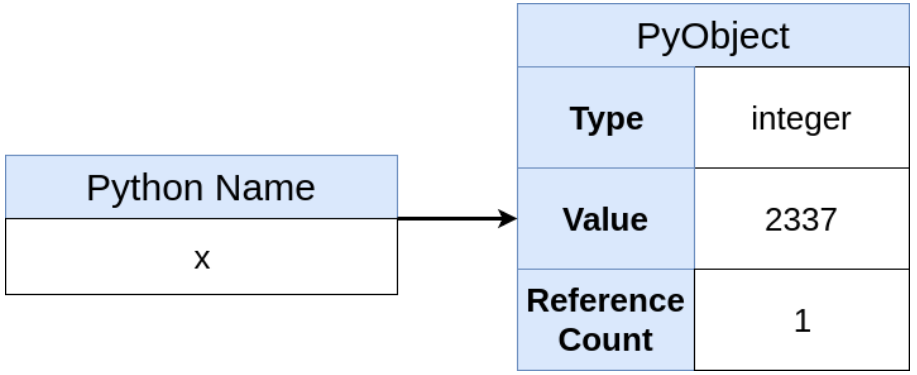
变量（对象）的三个关键属性

- **标识**（identity）用于唯一地表示一个对象， 通常对应对象在计算机内存中的位置。 **id(obj)**
- **类型**（type）用于标识对象所属的数据类型，数据类型用于限定对象的取值范围以及允许执行的操作。**type(obj)**
- **值**（value）用于表示对象的数据类型的值。 **obj**

变量赋值

变量的值并不是直接存储在变量里， 而是以对象的形式存储在内存某地址中。 我们可以说变量指向那个对象。 因此， Python变量里存放的实际是对象的位置信息（内存地址）。这种通过地址间接访问对象数据的方式，称为引用。

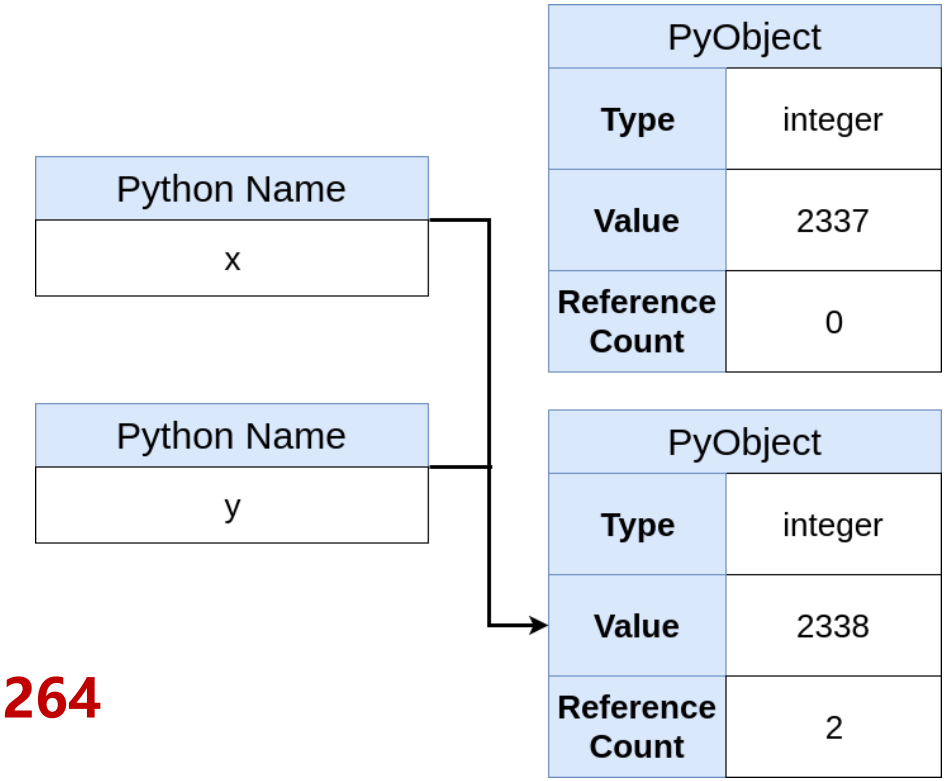




变量赋值

x 首次赋值后引用相应的数据对象，在执行加一操作后则引用新的数据对象，y = x 使y和x指向同一个对象。

```
>>> x = 2337
>>> id (x)
>>> 1418729158384
>>> y = x = x + 1
>>> print ( id(x), id(y) )
140730519193264 140730519193264
>>> print ( type(x), type(y) )
<class 'int’  > <class 'int'>
```



变量赋值

赋值语句并不返回值（**与C语言不同**），因此下边这条语句是不合法的

```
x = (y = y +1)
```

支持多重赋值，比如：

```
x = y = z = 5
```

支持多个变量同时赋值，或者叫元组赋值

```
x , y , z = 5 , 10 , 15
```

Q：变量同时赋值

```
a , b , c = 1, 1.5, 'a'
```

a、b、c的类型分别是什么?

```
a , b = b , a
a , b , c = b , c , a
```

a、b、c的值是什么？类型是什么?

基本数据类型

Python中有六个标准的数据类型：

- **Numbers（数字类型）**
- **Strings（字符串类型）**
- Tuples（元组类型）
- Lists（列表类型）
- Dictionaries（字典类型）
- Sets（集合类型）

基本数据类型：数字类型

包括**整型、浮点型和复数**三种，**布尔类型**作为**整型**的一个特殊子类型。

整型	浮点型	复数类型
10	10.0	5+6j

整型(int)

- ✓有“**无穷**”精度
- ✓可以使用多种进制

```
0b1010 # 二进制
0o12 # 八进制
10 # 十进制
0xA # 十六进制
```

布尔型(bool)

- ✓表示及运算与布尔代数完全一致

```
>>> 3 < 5
True
```

基本数据类型：数字类型

浮点型（float）

IEEE 754浮点数标准

- ✓ 所有浮点数必须带有小数部分，小数部分可以是0；
- ✓ 十进制表示和科学计数法表示；
- ✓ 数值范围存在限制，小数精度也存在限制；
- ✓ 2个浮点数一般不可以直接比较相等。

```
4.          # 十进制形式表示，相当于4. 0
. 5         # 十进制形式表示，相当于0. 5
-2.7315e2   # 科学计数法表示，相当于 -2.7315x10^2
```

复数型（Complex）

- ✓ 与数学中的复数概念一致，由实数部分和虚数部分构成；
- ✓ 复数的虚数部分通过后缀”J”或”j”来表示；
- ✓ 实部和虚部的数值都是浮点型；

```
num_one = 3 + 2j            # 直接创建
num_two = complex(3, 2)     # 通过内置函数创建
```

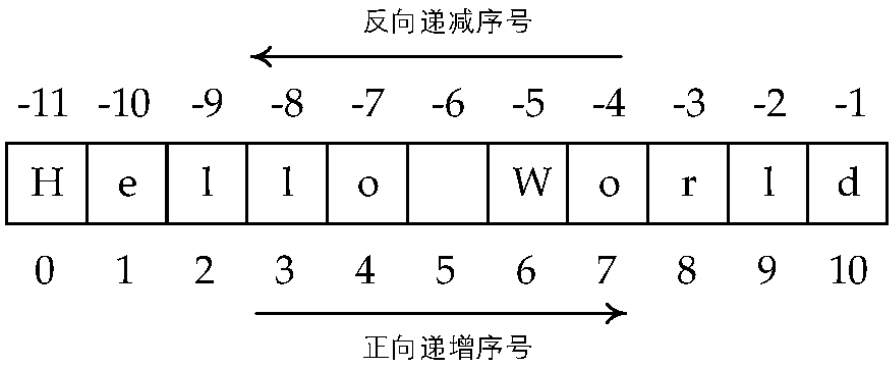
- ✓ 对于复数z，可以通过z.real和z.imag分别获得它的实数部分和虚数部分

基本数据类型：字符串类型

以单引号或双引号或三个引号括起来的任意文本；

用\做转义字符；用r前缀表示不转义

本质上是一个字符序列，支持双向索引。



r’ \n\t’ #不转义

R” C:\PythonPractice\nHelloPython.py”

常用转义符

- \\ 反斜杠符号
- \' 单引号
- \" 双引号
- \a 响铃
- \b 退格
- \f 进纸符或者叫换页符
- \n 换行
- \r 回车
- \t 横向制表符
- \v 纵向制表符
- \ooo 8进制数ooo表示的字符，数字可以是1到3个
- \xhh 16进制数hh表示的字符，数字必须是2个

基本数据类型：字符串类型

定义字符串时单引号与双引号可以嵌套使用，需要注意的是， 在单引号表示的字符串中可以嵌套双引号， 但是不允许嵌套单引号； 使用双引号表示的字符串中， 允许嵌入单引号， 但不允许包含双引号。

单引号	双引号	三引号
'He said "How are you"'	"It's my book"	""" His name's "T" bag """
当字符串中含有双引号时，最好使用单引号作为界定符	当字符串中含有单引号时，最好使用双引号作为界定符	常用于多行字符串，可以包含单双引号

基本数据类型：字符串类型

可以通过+或*进行**连接**。

```
>>> "Hello" + "World"
'HelloWorld'
>>> "Hello" * 3
'HelloHelloHello'
```

支持**“切片”**操作，切片的基本格式为

```
<string>[<start>:<end>]
```

常用操作

字符串属于不可变序列类型，也就是说**不允许通过索引改变字符串的值。**

```
s[3] = “H “ #错误的用法
```

可以通过+或**replace**函数。

```
s=s[:6] + ‘BUPT’
s=s.replace( ‘o’, ‘O’) #用O替换o
```

基本数据类型：字符串的格式化

使用字符串**格式运算符（%）**；（不再改进）

使用字符串的**format方法**;

f-string，Formatted String Literals。（推荐）

format方法的基本用法：

string.format(arg1,arg2...)

```
>>> "{}是一种程序设计语言，它目前的版本是{}".format("Python",3.7)
'Python是一种程序设计语言，它目前的版本是3.7。'
```

字符串的格式化　　**{:格式控制标记}.format(string)**

格式控制标记的基本格式：

[[填充]对齐] [符号][#][0][宽度][分隔符][.精度][类型]

标记类型	说明
填充	用于填充的单个字符
对齐	< 左对齐；> 右对齐；^ 居中对齐；= 仅对数字有效，将填充字符放到符号与数字间，例如 +0001234
符号	仅对数字有效，+ 正数加正号，负数加负号；- 正数不变，负数加负号；空格，正数加空格，负数加负号；
宽度	数字，为输出宽度
分隔符	,或-，数字的分隔符
精度	浮点数小数的精度或字符串的最大输出长度
类型	b、c、d、e、E、f、F、g、G、n、o、s、x、X、%

基本数据类型：字符串的格式化

语法描述

```
replacement_field ::= "{" [field_name] ["!" conversion] [":" format_spec] "}"
field_name ::= arg_name ("." attribute_name | "[" element_index "]")*
arg_name ::= [identifier | digit+]
attribute_name ::= identifier
element_index ::= digit+ | index_string
index_string ::= <any source character except "]"> +
conversion ::= "r" | "s" | "a"
format_spec ::= <described in the next section>
```

```
format_spec ::=
[[fill]align][sign][#][0][width][grouping_option][.precision][type]
fill ::= <any character>
align ::= "<" | ">" | "=" | "^"
sign ::= "+" | "-" | " "
width ::= digit+
grouping_option ::= "_" | ","
precision ::= digit+
type ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" |
"s" | "x" | "X" | "%"
```

基本数据类型：字符串的格式化

f-string

格式化字符串 Formatted String Literals

f-string的基本用法：

```
>>> f'{7 * 23}'
'161'
>>> name = 'Zhang'
>>> age = 19
>>> f'My name is {name}, and I am {age} years old.'
'My name is Zhang, and I am 19 years old.'
```

参考网址：https://www.python.org/dev/peps/pep-0498/

基本数据类型：字符串的其他内置函数

Python对于字符串类型提供了丰富的内置函数方便文本处理，如：

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> 'Hello world'.title()
'Hello World'
```

具体可以查阅Python官方文档

- ✓ 大小写转换函数
- ✓ 查找替换函数
- ✓ 字符判断函数
- ✓ 字符串头尾判断函数
- ✓ 计算函数
- ✓ 字符串拆分与合并

基本数据类型：空值

只有一个值就是**None**；

不支持任何运算，没有任何内置方法，也没什么有用的属性，

它的布尔值总是False；

```
>>> type(None)
<class 'NoneType'>
```

基本数据类型：类型转换

Python是**强类型**语言。 当一个变量被赋值为一个对象后，这个对象的类型就固定了，不能隐式转换成另一种类型。因此Python提供了类型转换内置函数。

- ✓不同数据类型进行混合运算时会隐式的进行数据类型的转换；
- ✓转换的原则为将所有的数据类型统一到最“大”的类型；
- ✓常用的数据类型转换函数：

bin()、oct()、hex()用来将整数转换为二进制、八进制和十六进制形式

int()和float()用来将其它类型的数据转换为整数和实数

str()用来将其任意类型参数转换为字符串

基本数据类型：类型转换

转换为整型：

int(x [,base])

int()函数将x转换为一个整数，x为字符串或数字，base进制数，默认为十进制。

```
>>> int(100.1)          #浮点转整数
100                     #返回结果
>>> int('01010101',2)   #二进制转换整数
85                      #返回结果
```

转换为浮点型：

float(x)

float()函数将x转换为一个浮点数，x为字符串或数字，没有参数的时默认返回0.0。

```
>>> float()             #空值转换
0.0                     #返回结果
>>> float(1)            #整数转浮点
1.0                     #返回结果
>>> float('120')        #字符转浮点
120.0                   #返回结果
```

基本数据类型：类型转换

转换为字符串类型：

str(x)

str() 函数将对象转化为适于人阅读的形式，返回值为对象的string类型。

```
>>> x = "今天是晴天"    #定义x
>>> str(x)              #对x进行转换
'今天是晴天'            #返回结果
```

转换为布尔型：

bool(x)

bool() 函数用于把给定参数转换为布尔类型，返回值为True或者False，在没有参数的情况下默认返回 False。

```
>>> bool()          #空置转布尔类型
False               #返回结果
>>> bool(0)         #整数0转布尔值
False               #返回结果
>>> bool(1)         #整数1转布尔值
True                #返回结果
>>> bool(100)       #整数100转布尔值
True                #返回结果
```

运算符与表达式：运算符

运算符是一些特殊的符号， 主要用于数学计算、 比较大小和逻辑运算等。Python的运算符主要包括算术运算符、 赋值运算符、 比较运算符、逻辑运算符以及位运算符。

优先级

结合性： 左结合或右结合

使用圆括号可以改变运算符的运算次序

操作数目 ： 可以分为单目 、双目和三目运算符

运算符与表达式：表达式

使用运算符将不同类型的数据按照一定的规则连接起来的式子，称为表达式

表达式由运算符和参与运算的数（操作数） 组成。

按照运算符的种类， 表达式可以分成： 算数表达式、关系表达式、 逻辑表达式、测试表达式等。

多种运算符混合运算形成复合表达式， 按照运算符的优先级和结合性依次进行运算。

很多运算对操作数的类型有要求， 例如 加法要求两个操作数类型一致，当操作数类型不一致时，可能发生隐式类型转换。

运算符与表达式：算数运算符

算术运算符包括+、 -、 *、 /、 //、 %和**， 都是双目运算符， 每个运算符可以与两个操作数组成一个表达式。

- ✓ 加法运算及乘法运算还可以用于字符串类型；
- ✓ 幂运算支持复数、实数及整数；
- ✓ 除法运算是数学意义上的除法，运算结果为浮点数；
- ✓ 整除运算可以用于实数和整数；
- ✓ 取余运算可用于实数和整数。

算术运算符优先级：

- 相同类型之间的数据运算， 算术运算符优先级由高到最低顺序排列如下：
  第一级： **
  第二级： *， /， %， //
  第三级： +， -
- 不同类型之间的数据运算， 会发生隐式类型转换。 转换规则是： 低类型向高类型转换。 可以进行算术运算的各种数据类型， 从低到高排列为：
  bool<int<float<complex

运算符与表达式：比较运算符

包括==、!=、<、>、<=、>=、is、is not，最大特点是可以连用，其含义与我们日常的理解一致。

```
1<= n<=10

'a' <= ch <= 'z'

3 < a > 1等价于3<a && a>1

1== a <=10

1== (a<=10)
```

常用字符的大小关系为：

空字符<空格<' 0' ~' 9' <' A' ~' Z' <' a' ~' z' <汉字

通过 == 运算符可以判断两个变量指向的对象值是否相同

通过 **is** 运算符可以判断两个变量是否指向同一对象

```
a=300
b=300
print(a==b)          # True
print(a is b)        # False
str1 = 'hello'
str2 = 'hello'
print(str1 is str2) )   # True
```

- ✓ 小整数对象[-5,256]是全局解释器范围内被重复使用，永远不会被GC回收。
- ✓ 同一个代码块中的不可变对象，只要值是相等的就不会重复创建新的对象。

运算符与表达式：逻辑运算符

- ✓ and、or、not分别代表与运算、或运算、非运算；优先级**not>and>or**
- ✓ and和or并不一定会返回True或False，而是得到最后一个被计算的表达式的值；运算符not一定会返回True或False。
- ✓ and和or具有惰性求值或叫逻辑短路的特点；
- ✓ 三目运算符语法：语句1 if 条件（比较）表达式 else 语句2

None、任何数值类型中的0、空字符串“”、空元组()、空列表[]、空字典{}的逻辑值都是False，其他对象均为True。

运算符与表达式：逻辑运算求值

- ✓若 **or** 的表达式中的对象逻辑值都为 **False**，则返回最后一个**False**
- ✓若 **and** 的表达式中的对象逻辑值都为**True**，则返回最后一个**True**
- ✓实际返回值是返回的那个逻辑值对应的对象值

```
# 若上下文中所有逻辑值均为True,则返回从左到右的最后一值！！而不一定返回
True
>>> 1 and 2 # 返回最后一个 2
2
>>> 1 and 2 and 3 and 4 # 返回最后一个 4
4
# 若上下文中所有逻辑值均为False,则返回从左到右的最后一值！！而不一定返回
False
>>> 0 or False # 返回最后一个 False
False
>>> '' or {} or () # 返回最后一个 ()
()
>>> 0 or [] or None # 无显式返回（返回最后一个 None）
```

运算符与表达式：逻辑短路

- ✓若 **or** 的左侧逻辑值为 **True**，则短路 **or** 后所有的表达式（**不管是 and 还是 or**）
- ✓若 **and** 的左侧逻辑值为 **False**，则短路其后所有 **and** 表达式，**直到有 or 出现**，接下来的逻辑运算忽略短路部分的表达式

```
# 从左到右依次演算，一旦遇到 bool值为 True 的对象，立即返回之，不再往后演算
>>> 1 or 2 or 3 or 0
1
# 从左到右依次演算，一旦遇到 bool值为 False 的对象，立即返回之，不再往后演算
>>> 1 and '' and {} and () and False
''
# 短路 and 和 or 之间的逻辑计算
>>> 1 and 0 and [] and None or 6
6
```

应用：求a和b中的较大者，赋值给c

```
a, b, c = 1, 2, 3
if a>b:
  c = a
else:
  c = b
```

```
c = a if a>b else b
c= [b, a][a > b]
c = (a>b and a or b)
c = (a>b and [a] or [b])[0]
```

C语言：c = a>b ? a : b

运算符与表达式：赋值运算符

- ✓ = 是主要的赋值运算符。
- ✓ 增强赋值运算符：+=、-=、*=、**=、/=、//=、%=。
- ✓ 如果用op表示算术运算符，则下面两个赋值操作是等价的：

  x op= y 与 x = x op y。如x+=5，与x=x+5。

如何记忆赋值与等号？与等号有关的时候都是两个符号：

==、!=、<=、>=：==，“等于”等于

运算符与表达式：运算符优先级汇总

- ✓ 按优先级顺序
- ✓ 小括号优先
- ✓ 同级运算符左结合
- ✓ 赋值运算右结合

不要图省事只依赖于优先级，合理使用括号！

优先级	类别	运算符	说明
最高	算术运算符	**	指数，幂
高	位运算符	+x,-x,~x	正取反，负取反，按位取反
	算术运算符	*,/,%,//	乘，除，取模，取整
	算术运算符	+,-	加，减
	位运算符	>>,<<	右移，左移运算符
	位运算符	&	按位与，集合并
	位运算符	^	按位异或，集合对称差
	位运算符	|	按位或，集合并
	比较运算符	<=,<,>,>=	小于等于，小于，大于，大于等于
	比较运算符	==,!=	等于，不等于
	赋值运算符	=,%=,/=,//=, -=,+=,*=,**=	赋值运算
	逻辑运算符	not	逻辑“非”
	逻辑运算符	and	逻辑“与”
低	逻辑运算符	or	逻辑“或”

基本输入输出

输入 input()

接收任意输入，返回字符串类型；

在获得用户输入之前可以包含一些提示性文字；

```
x = input("请输入：")
a , b = map(int, input().split())
data = sorted(list(map(int, input().split())))
data = [input() for i in range(n)]
```

输入的字符串可以通过各种字符串的处理方法进行处理得到需要的数据。

输出 print()

语法格式为：

print(*objects, sep=’ ’, end=’\n’, file=sys.stdout, flush=False)

objects为需要输出的内容；sep用于指定数据之间的分隔符；end用于指定输出的结尾；file用于指定输出位置；flush确定输出是否使用缓存。

结合字符串的格式化可以得到各种格式化的输出。

小结

- 标识符、变量和赋值

  动态类型、对象引用

- 简单数据类型

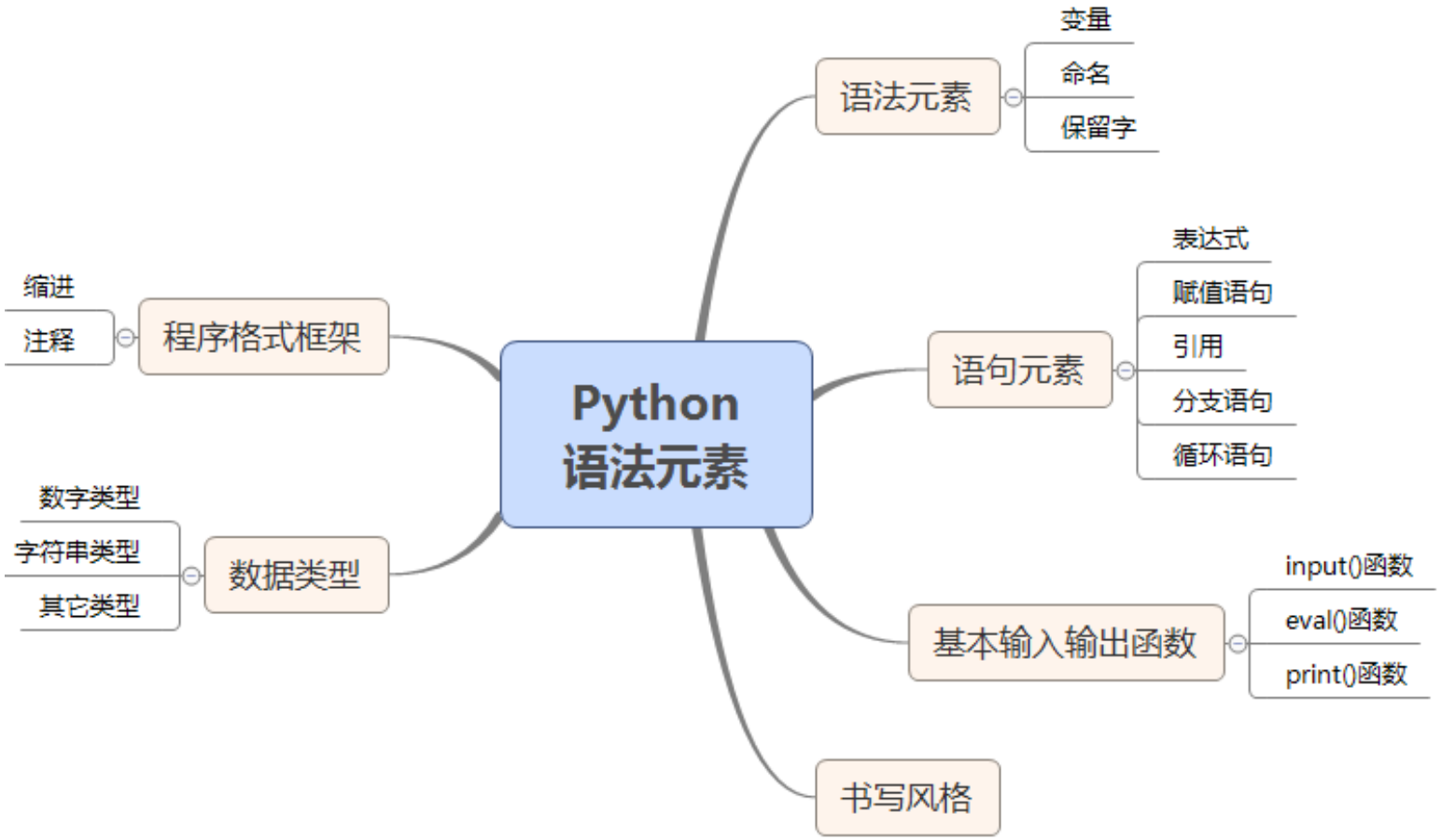
  强类型及类型转换

- 运算符和表达式

  优先级、逻辑运算

- 基本输入输出

  格式化



基础语法

- 语法的表示
- 标识符、变量及其赋值
- 基本数据类型
- 运算符与表达式
- 基本输入输出

谢谢