

# Python编程及人工智能应用

## 第二章 Python语言基础

<https://bolei-zhang.github.io/course/python-ai.html>

<https://www.icourse163.org/spoc/course/NJUPT-1464077167>

- Python基本语法
  - 变量和数据类型
  - 运算符和表达式
  - 字符串
  - 流程控制
- Python组合数据类型
- Python函数
- 异常处理和文件操作
- Python面向对象基础
- 数值计算库NumPy

# 为什么命名为Python

- 在着手编写 Python 实现的时候，Guido van Rossum 同时还阅读了刚出版的 "Monty Python 的飞行马戏团" 剧本，这是一部自 1970 年代开始播出的 BBC 系列喜剧。Van Rossum 觉得他需要选择一个简短、独特而又略显神秘的名字，于是他决定将这个新语言命名为 Python。





# 交互模式vs脚本模式

- 交互模式

- 直接在命令行输入python或python3，每次回车之后运行一行代码

- 脚本模式

- 编辑后缀为.py的文件
- 然后通过 python [文件名].py运行
  
- Windows推荐用windows terminal
- MAC推荐用Iterm
- 自学一些简单的Linux shell命令
  - ls, pwd, cd, which, ps...

```
>>> L = []
>>> dir(L)
['_add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
 '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear',
 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
 'reverse', 'sort']
>>> [d for d in dir(L) if '_' not in d]
['append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 're

>>> help(L.append)
Help on built-in function append:

append(...)
    L.append(object) -> None -- append object to end

>>> L.append(1)
>>> L
[1]
```

## •数据类型

- Python的变量不用声明，对象是储存在内存中的实体。但我们并不能直接接触到该对象。我们在程序中写的对象名，只是指向这一对象的引用(reference)。
- Python对象的类型和内存都是在运行时确定的

## •编译

- 当Python程序运行时，编译的结果则是保存在位于内存中的PyCodeObject中，当Python程序运行结束时，Python解释器则将PyCodeObject写回到pyc文件中。
- 当Python程序第二次运行时，首先程序会在硬盘中寻找对应的pyc文件，如果找到，则直接载入，否则就重复上面的过程。

## •指针

- Python函数的参数是引用传递（除非对于不可变类型）

## •内存管理

- Python采用了类似Windows内核对象一样的方式来对内存进行管理。每一个对象，都维护这一个对指向该对象的引用的计数
- 当内存中有不再使用的部分时，垃圾收集器就会把他们清理掉。它会去检查那些引用计数为0的对象，然后清除其在内存的空间。

- 使用**缩进**来表示程序的层次结构。不同于C语言，**Python变量可不声明直接使用**
  - 同一层次代码块，缩进所包含空格数量必须一致
    - #代码2.1 接收用户从键盘上输入的整数，并判别其奇偶性
    - `num = int(input( "请您输入一个整数：" ))`
    - `if num%2==0:`
      - `print(num,"是一个偶数")`
      - `print("这里执行的是程序中的第1个分支")`
    - `else:`
      - `print(num,"是一个奇数")`
      - `print("这里执行的是程序中的第2个分支")`
  - 输入函数`input()`，从键盘输入一个字符串
    - `int()`函数用于把字符串转换成整数
  - 输出函数`print()`，输出后自动换行



# 为什么Python使用缩进

Guido van Rossum 认为使用缩进进行分组非常优雅，并且大大提高了普通Python程序的清晰度。大多数人在一段时间后就会学会并喜欢上这个功能。

由于没有开始/结束括号，因此解析器感知的分组与人类读者之间不会存在分歧。偶尔C程序员会遇到像这样的代码片段：

```
if (x <= y)
    x++;
    y--;
z++;
```

如果条件为真，则只执行 `x++` 语句，但缩进会使你认为情况并非如此。即使是经验丰富的C程序员有时会长时间盯着它，想知道为什么即使 `x > y`，`y` 也在减少。

因为没有开始/结束括号，所以Python不太容易发生编码式冲突。在C中，括号可以放到许多不同的位置。如果您习惯于阅读和编写使用一种风格的代码，那么在阅读（或被要求编写）另一种风格时，您至少会感到有些不安。

许多编码风格将开始/结束括号单独放在一行上。这使得程序相当长，浪费了宝贵的屏幕空间，使得更难以对程序进行全面的了解。理想情况下，函数应该适合一个屏幕（例如，20--30行）。20行Python可以完成比20行C更多的工作。这不仅仅是由于缺少开始/结束括号 -- 缺少声明和高级数据类型也是其中的原因 -- 但缩进基于语法肯定有帮助。

- Python 所有数据都由**对象**或**对象间关系**来表示
- Python常见的内置标准类型
  - **整型数字 ( int )**
  - **浮点型数字 ( float )**
  - **逻辑值 ( bool )**
  - **字符串 ( str ) \***
  - **复数型数字 ( complex )**
  - **元组 ( tuple )、不可变集合 ( frozenset )**
  - **列表 ( list )、可变集合 ( set )、字典 ( dict )**

## 不可变对象

- 可 hash (不可变长度)
- 不支持新增
- 不支持删除
- 不支持修改
- 支持查询

## 可变对象

- 不可 hash (可变长度)
- 支持新增
- 支持删除
- 支持修改
- 支持查询

#代码2.2 在屏幕上输出各种对象的类型

```
print("100 is", type(100))
print("3.14 is", type(3.14))
print("5+2j is", type(5+2j))
print("True and False are", type(False))
print("\'I love Python.\' is", type("I love Python."))
print("[1,2,3] is", type([1,2,3]))
print("(1,2,3) is", type((1,2,3)))
print("{1,2,3} is", type({1,2,3}))
print("frozenset({1,2,3}) is", (frozenset({1,2,3})))
print("{'name':'Tom','age':18} is",
type({'name':'Tom','age':18}))
```

代码2.2运行后输出如下:

```
100 is <class 'int'>
3.14 is <class 'float'>
5+2j is <class 'complex'>
True and False are <class 'bool'>
'I love Python.' is <class 'str'>
[1,2,3] is <class 'list'>
(1,2,3) is <class 'tuple'>
{1,2,3} is <class 'set'>
frozenset({1,2,3}) is <class 'frozenset'>
{'name':'Tom','age':18} is <class 'dict'>
```

- type()是Python的内置函数，返回对象所属类型

## •标识符的命名规则

- 由大写和小写字母、下划线 \_ 以及数字组成
- 不可以数字打头，可以包含Unicode字符（中文）
- 不可以和Python中的关键字重复

## •Python关键字

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

- 赋值语句可以将**变量**和**对象**进行**关联**

```
#代码2.3 使用赋值语句建立变量和对象之间的关联
#这是使用科学记数法表示浮点数0.0178的方法
var = 1.78E-2
print(var, type(var))
#复数的表示中，带有后缀j或者J的部分为虚部
var = 3+7j
print(var, type(var))
#字符串数据既可以用单引号，也可以用双引号进行定义
var = "人生苦短，我用Python"
print(var, type(var))
```

```
运行代码2.3输出如下
0.0178 <class 'float'>
(3+7j) <class 'complex'>
人生苦短，我用Python <class 'str'>
```

```
#代码2.4 赋值语句的连续赋值和多重赋值
#用于将不同的变量关联至同一个对象
x = y = z = 100
print(x, y, z)
#用于将不同的变量关联至不同的对象
a, b, c = 7, 8, 9
print(a, b, c)
```

```
运行代码2.4后输出如下
100 100 100
7 8 9
```

# Python运算符和表达式

- 表达式可以由多个**运算对象**和**运算符**构成
- Python按照运算符的**优先级**求解表达式的值
  - Python常见运算符（从最低优先级到最高优先级）；Python没有三目运算符，但是可以用if else

运算符	运算符描述
if -- else	条件运算符
or	逻辑或运算
and	逻辑与运算
not x	逻辑非运算
in、not in、is、is not、<、<=、>、>=、!=、==	比较运算
	按位或运算
^	按位异或运算
&	按位与运算
<<、>>	移位运算
+、-	算术运算符：加、减
*、/、//、%	算术运算符：乘、除、整除、取模（求余数）
+x、-x、~x	单操作数运算符：正、负、按位非运算
**	乘方运算符

# Python运算符和表达式



## #代码2.6 表达式举例

<code>print(1+2*3-4)</code>	#由于乘号的优先级高于加号和减号，所以2*3会优先计算
<code>print(2*3**3)</code>	#乘方运算的优先级比乘法运算还要高
<code>a, b = 10, 20</code>	
<code>print(a if a&gt;b else b)</code>	#输出变量a和b中数值较大的那一个
<code>print(True+1)</code>	#运算过程中，True和False可以被自动转换成1和0使用
<code>print(3&gt;2&gt;1)</code>	#连续的比较运算，其结果与表达式3>2 and 2>1等价
<code>print((3&gt;2)&gt;1)</code>	#这条表达式计算的是子表达式3>2的值是否大于1，即True>1

运行代码2.6后输出如下：

```
3
54
20
2
True
False
```

- 字符串类型的对象使用一对**单引号**、一对**双引号**或者一对**三引号**进行定义
  - 使用三引号定义字符串的时候可以包含换行符

```
s = '''Hello  
World'''
```

- Python字符串转义字符

转义字符	含义	转义字符	含义
\'	单引号	\t	水平制表符
\"	双引号	\v	垂直制表符
\\	字符“\”本身	\r	回车符
\a	响铃	\f	换页符
\b	退格符	\ooo	以最多3位的八进制数作为编码值对应的字符
\n	换行符	\xhh	以必须为2位的十六进制数作为编码值对应的字符

# Python字符串举例



#代码2.7 定义字符串的几种方法

```
str1 = '单引号可以用来定义字符串'  
str2 = "双引号也可以用来定义字符串"  
print(str1, str2)  
str3 = """三引号定义的字符串可以直接换行  
这是字符串的第二行  
这是字符串的第三行"""  
print(str3)  
print("转义字符\n表示换行符")  
print(r"转义字符\n表示换行符")  
#格式标记.2f表示保留小数点后2位小数  
print(f"半径为5的圆的面积是{3.14*5**2:.2f}")
```

运行代码2.7后输出如下：

```
单引号可以用来定义字符串 双引号也可以用来定义字符串  
三引号定义的字符串可以直接换行  
这是字符串的第二行  
这是字符串的第三行  
转义字符  
表示换行符  
转义字符\n表示换行符  
半径为5的圆的面积是78.50
```

Python字符串是不可变的

# Python字符串不可变



有几个优点。

一个是性能：知道字符串是不可变的，意味着我们可以在创建时为它分配空间，并且存储需求是固定不变的。这也是元组和列表之间区别的原因之一。

另一个优点是，Python 中的字符串被视为与数字一样“基本”。任何动作都不会将值 8 更改为其他值，在 Python 中，任何动作都不会将字符串 "8" 更改为其他值。

```
>>> a = 'Hello World!'
>>> a[0] = 'Y'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> a += '?'
>>> a
'Hello World!?'
>>> a.replace('H', 'Y')
'Yello World!?'
>>> a
'Hello World!?'
```

# Python字符串运算



- f-string，亦称为格式化字符串常量，从Python3.6新引入，目的是使格式化字符串操作更加简便
- f-string在形式上是以f或F修饰符引领的字符串： **格式：f" {表达式}"**

```
#代码2.8 与字符串有关的运算
#字符串之间用加法运算连接，结果为连接后的字符串
print("ABCD"+"1234")
#字符串和整数进行乘法运算，结果为字符串复制多遍并连接
print("Hi"*3)
str1 = "I love Python!"
#获取字符串str1中的首个字符
print(f"str1[0] = {str1[0]}")
#获取字符串str1的最后一个字符
print(f" str1[-1] = {str1[-1]}")
#获取字符串中索引号从2开始到6之前结束的子字符串，即"love"
print(f" str1[2:6] = {str1[2:6]}")
```

运行代码2.8后输出如下：

```
ABCD1234
HiHiHi
str1[0] = 'I'
str1[-1] = '!'
str1[2:6] = 'love'
```

# Python字符串的常用函数



## • 字符串对象有一系列已定义好的函数可直接使用

#代码2.9 字符串对象的常用方法

```
str1 = "i have an apple."
```

```
print(f" str1.capitalize() = {str1.capitalize()}") #用于将字符串的首字母大写
```

```
print(f" str1.title() = {str1.title()}") #用于将字符串中每个单词的首字母大写
```

```
print(f" str1.count('a') = {str1.count('a')}") #用于统计指定的字符在字符串中出现的次数
```

```
print(f" str1.startswith('i am') = {str1.startswith('i am')}") #用于判定字符串是否以指定的参数开始
```

```
print(f" str1.endswith('apple.') = {str1.endswith('apple.')}") #用于判定字符串是否以指定的参数结尾
```

```
print(f" str1.find('apple') = {str1.find('apple')}") #用于查找指定的子字符串并返回其起始位置
```

```
print(f" str1.find('pear') = {str1.find('pear')}") #若无法找到指定的子字符串，find()方法会返回-1
```

```
print(f" str1.split() = {str1.split()}") #对字符串进行切割，默认为所有的空字符，包括空格、换行(\n)、制表符(\t)等。作为分隔符
```

```
print(f" ','.join(['a','b','c']) = {','.join(['a','b','c'])}") #用指定的字符串将若干字符串类型的对象进行连接
```

```
print(f" 'ABcd'.upper() = {'ABcd'.upper()}") #将字符串中的字母转换成大写
```

```
print(f" 'ABcd'.lower() = {'ABcd'.lower()}") #将字符串中的字母转换成小写
```

```
print(f" ' ABcd '.strip() = {' ABcd '.strip()}") #删除字符串前后的特殊字符，比如空格和换行符
```

运行代码2.9后的输出结果：

```
str1.capitalize() = 'I have an apple.'
```

```
str1.title() = 'I Have An Apple.'
```

```
str1.count('a') = 3
```

```
str1.startswith('i am') = False
```

```
str1.endswith('apple.') = True
```

```
str1.find('apple') = 10
```

```
str1.find('pear') = -1
```

```
str1.split() = ['i', 'have', 'an', 'apple.']
```

```
' , '.join(['a','b','c']) = 'a,b,c'
```

```
'ABcd'.upper() = 'ABCD'
```

```
'ABcd'.lower() = 'abcd'
```

```
' ABcd '.strip() = 'ABcd'
```

- 用 **if...elif...else** 来构造选择结构的程序代码
- python没有switch语句

```
#代码2.10 将百分制成绩转换成五级制成绩
score = float(input("请输入一个百分制成绩："))
if score >= 90:
    print(f"成绩{score}对应的五级制成绩是优秀")
elif score >= 80:
    print(f"成绩{score}对应的五级制成绩是良好")
elif score >= 70:
    print(f"成绩{score}对应的五级制成绩是中等")
elif score >= 60:
    print(f"成绩{score}对应的五级制成绩是及格")
else:
    print(f"成绩{score}对应的五级制成绩是不及格")
```

运行代码2.10后的输出结果：

```
请输入一个百分制成绩：95
成绩95.0对应的五级制成绩是优秀
```

## 为什么Python中没有switch或case语句？

你可以通过一系列 `if... elif... elif... else` 轻松完成这项工作。对于switch语句语法已经有了一些建议，但尚未就是否以及如何进行范围测试达成共识。有关完整的详细信息和当前状态，请参阅 [PEP 275](#)。

对于需要从大量可能性中进行选择的情况，可以创建一个字典，将case 值映射到要调用的函数。例如：

```
def function_1(...):
    ...

functions = {'a': function_1,
            'b': function_2,
            'c': self.method_1, ...}

func = functions[value]
func()
```

- 通过组合多个if...elif...else结构进行嵌套使用

```
#代码2.11 判断从键盘上输入的三个整数构成的三角形形状
a,b,c = input("请输入组成三条边，用空格分隔：").split()
a,b,c = int(a), int(b), int(c)
a,c,b = min(a,b,c), max(a,b,c), a+b+c-min(a,b,c)-max(a,c,b)
if a+b<=c:
    print("输入的整数无法构成三角形")
else:
    if a==b==c:
        print("输入的整数构成一个等边三角形")
    elif a==b:
        print("输入的整数构成一个等腰三角形")
    elif a**2+b**2==c**2:
        print("输入的整数构成一个直角三角形")
    else:
        print("输入的整数构成一个普通三角形")
```

## 运行代码2.11后的输出结果：

请输入组成三角形三条边的整数，用空格分隔：1 1 2  
输入的整数无法构成三角形

请输入组成三角形三条边的整数，用空格分隔：3 3 3  
输入的整数构成一个等边三角形

请输入组成三角形三条边的整数，用空格分隔：3 3 5  
输入的整数构成一个等腰三角形

请输入组成三角形三条边的整数，用空格分隔：3 4 5  
输入的整数构成一个直角三角形

请输入组成三角形三条边的整数，用空格分隔：5 6 7

- 用关键字 **while** 或者 **for** 可以用来构造循环结构
  - while 语句实现累加
  - for 语句实现累加

#代码2.12 使用while循环完成1至100所有整数求和

```
n = 1
s = 0
while n <= 100:
    s += n      #与s = s + n等价
    n += 1     #与n = n + 1等价
print("1到100所有整数的和是 :", s)
```

#代码2.13 使用for循环完成1至100所有整数求和

```
s = 0
for n in range(1, 101):
    s += n
print("1到100所有整数的和是 :", s)
```

- range(1,101)所表示的从1到101以内（不含101）的所有整数

- 如果range()函数只有一个参数，则该参数表示返回结果的终止值（不含）
- 如果range()函数有两个参数，则第1个参数表示返回结果的起始值，第2个参数表示返回结果的终止值（不含）
- 如果range()函数有三个参数，则第1个参数表示返回结果的起始值，第2个参数表示返回结果的终止值（不含），第3个参数表示每次步进的增量值

```
#代码2.14 求100以内所有奇数的和  
#sum()函数的功能为对其参数进行求和  
s = sum(range(1,100,2))  
print("100以内所有奇数的和是：",s)
```

- **break**语句会终结最近的外层循环，如果循环有可选的else子句，也会跳过该子句
- **continue**语句会终止当前循环中剩下的语句，并继续执行最近的外层循环的下一个轮次

```
#代码2.15 输出从小到大排列的第100个素数
count = 0
num = 2
while True:
    #下方的for循环用于判断num是否为素数，如果是
    #则计数器count加1
    for i in range(2,num):
        if num%i==0:
            break
    else:
        count += 1
```

```
#如果计数器小于100，num加1，且终止当前循环中剩下的
#语句，并继续执行循环的下一个轮次
if count<100:
    num += 1
    continue
#剩下的代码只有在上方if不成立的时候才会被执行到，
#即计数器count为100的情况下
print(num,"是从小到大排列的第100个素数")
break #输出完毕后，使用break语句终结外
层的while循环
```

# 一起读代码



```
1 """
2 Approximates the area under the curve using the trapezoidal rule
3 """
4 from __future__ import annotations
5
6 from collections.abc import Callable
7
8
9 def trapezoidal_area(
10     fnc: Callable[[int | float], int | float],
11     x_start: int | float,
12     x_end: int | float,
13     steps: int = 100,
14 ) -> float:
15     """
16     Treats curve as a collection of linear lines and sums the area of the
17     trapezium shape they form
18     :param fnc: a function which defines a curve
19     :param x_start: left end point to indicate the start of line segment
20     :param x_end: right end point to indicate end of line segment
21     :param steps: an accuracy gauge; more steps increases the accuracy
22     :return: a float representing the length of the curve
23
24     >>> def f(x):
25     ...     return 5
26     >>> f"{trapezoidal_area(f, 12.0, 14.0, 1000):.3f}"
27     '10.000'
28     >>> def f(x):
29     ...     return 9*x**2
```

```
30 >>> f"{trapezoidal_area(f, -4.0, 0, 10000):.4f}"
31 '192.0000'
32 >>> f"{trapezoidal_area(f, -4.0, 4.0, 10000):.4f}"
33 '384.0000'
34 """
35 x1 = x_start
36 fx1 = fnc(x_start)
37 area = 0.0
38 for _ in range(steps):
39     # Approximates small segments of curve as linear and solve
40     # for trapezoidal area
41     x2 = (x_end - x_start) / steps + x1
42     fx2 = fnc(x2)
43     area += abs(fx2 + fx1) * (x2 - x1) / 2
44     # Increment step
45     x1 = x2
46     fx1 = fx2
47 return area
48
```

[https://github.com/TheAlgorithms/Python/blob/master/maths/area\\_under\\_curve.py](https://github.com/TheAlgorithms/Python/blob/master/maths/area_under_curve.py)

## 1、无下划线变量

xx: 无下划线变量为公有变量

## 2、前面单下划线

xx: 前置单下划线, 又称口头私有变量, 私有化属性或方法的一种, 一般来讲, 变量名 xx 被看作是“私有的”, 在模块或类外不可以使用。当变量是私有的时候, 用 xx 来表示变量是很好的习惯。类对象和子类可以访问, 这并不能完全做到真正的私有, 只是约定俗成的而已, 这样写表示不希望这个变量在外部被直接调用

## 3、前面双下划线

xx: 前置双下划线, 私有化属性或方法, 只有内部可以访问, 外部不能访问。

## 4、前后都有双下划线

xx: 以双下划线开头, 并且以双下划线结尾的, 是特殊变量 (这就是在python中强大的魔法方法), 特殊变量是可以直接访问的, 对于普通的变量应当避免这种命名风格。

## 5、后置下划线

xx\_: 后置单下划线, 用于避免与Python关键词的冲突

# Python编程及人工智能应用

## 第二章 Python语言基础

<https://bolei-zhang.github.io/course/python-ai.html>



**课堂码：8CUS9W**

邀请学生使用中国大学MOOC APP 或微信扫一扫

- Python基本语法
  - 变量和数据类型
  - 运算符和表达式
  - 字符串
  - 流程控制
- Python组合数据类型
- Python函数
- 异常处理和文件操作
- Python面向对象基础
- 数值计算库NumPy



# 组合数据类型——列表

- 列表 ( list ) 是一种序列数据类型，是一种可变数据类型，内部元素可以任意增删和修改
- 列表对象的定义和基本运算如下：

```
##代码2.16 列表对象的定义和基本运算
```

```
lst1 = [] #定义一个空列表
```

```
#len()函数可以返回参数对象中包含元素的数量
```

```
print(f"len(lst1) = {len(lst1)}")
```

```
#list()函数可以将参数对象转换成列表
```

```
lst1 = list("Python!")
```

```
#列表也是序列类型对象，所以支持索引和切片
```

```
print(f" lst1[2] = {lst1[2]}, lst1[-2] = {lst1[-2]}")
```

```
print(f" lst1[2:-2] = {lst1[2:-2]}")
```

```
lst1 = [1,2,3]
```

```
lst2 = [1,2,3]
```

```
#通过id()函数我们可以取得对象的身份标识
```

```
print(f"{id(lst1) }, {id(lst2) }")
```

```
print("lst1和lst2是否相等：", lst1==lst2) # True
```

```
print("lst1和lst2是否相同：", lst1 is lst2) # False
```

```
#不同的变量与同一个列表对象关联
```

```
lst1 = lst2 = [1,2,3]
```

```
#无论操作变量lst1或者lst2，其实都是在操作同一个列表
```

```
lst1[0] = 100
```

# 列表的常用方法

- 列表对象具有一系列定义好的方法可以直接使用

#代码2.17 列表对象的常用方法

```
lst1 = list("Python")
```

```
lst1.append("666") #追加元素到列表中
```

```
lst1.insert(1, "is") #插入元素到列表中
```

```
lst1.remove("i") #从列表中移除指定的元素
```

```
#从指定位置弹出列表元素
```

```
# 默认弹出列表的最后一个元素
```

```
print(f' lst1.pop(0) = {lst1.pop(0)}' )
```

```
#将参数转换成列表元素后
```

```
#连接到原列表的末尾
```

```
lst1.extend([1, 2, 3])
```

```
#对列表元素按照ASCII码从小到大的顺序进行排序
```

```
lst1.sort()
```

```
#在列表中对指定的内容进行计数
```

```
print("'o'在lst1中出现了", lst1.count("o"), "次")
```

```
#在列表中查找指定的内容，返回其索引值
```

```
print("'t'在lst1中的索引值是：", lst1.index("t"))
```

```
lst1 = [1,2,3]
```

```
#通过copy()方法可以复制原列表对象
```

```
#得到一个新的列表对象
```

```
lst2 = lst1.copy()
```

```
lst2[0] = 100
```

```
print(f'{lst1 = }, {lst2 = }')
```

```
>>> L = []
>>> dir(L)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
 '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear',
 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
 'reverse', 'sort']
>>> [d for d in dir(L) if '_' not in d]
['append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 're

>>> help(L.append)
Help on built-in function append:

append(...)
    L.append(object) -> None -- append object to end

>>> L.append(1)
>>> L
[1]
```

- 列表推导式将列表元素的生成规则放置在一对方括号内完成列表的创建

#代码2.18 列表推导式的使用

```
lst1 = [n for n in range(1,10) if n%2==1] #创建列表包含10以内的奇数
```

```
lst2 = [5*n for n in range(5)] #创建等差数列构成的列表
```

```
lst3 = [3**n for n in range(5)] #创建等比数列构成的列表
```

#判断元素是否在数组里

```
5 in lst2
```

- Python列表是数组还是链表？

## 列表是如何在CPython中实现的？

CPython的列表实际上是可变长度的数组，而不是lisp风格的链表。该实现使用对其他对象的引用的连续数组，并在列表头结构中保留指向该数组和数组长度的指针。

这使得索引列表 `a[i]` 的操作成本与列表的大小或索引的值无关。

当添加或插入项时，将调整引用数组的大小。并采用了一些巧妙的方法来提高重复添加项的性能；当数组必须增长时，会分配一些额外的空间，以便在接下来的几次中不需要实际调整大小。

- 栈

- 先进后出，后进先出
- append()
- pop()

- 队列

- 先进先出
- insert(0, [element])
- pop()

- append和insert，哪个效率更高，为什么？

# 栈 or 队列



- `l = []`
- `for i in range(1000000):`
- `l.append(i)`

- `l = []`
- `for i in range(1000000):`
- `l.insert(0, i)`

- 元组 ( tuple ) 也是一种序列数据类型，但是它是一种**不可变**数据类型，其元素不可增删改

```
#代码2.19 元组对象的定义和基本运算
```

```
#定义空元组、一元组、二元组
```

```
tup1,tup2,tup3 = tuple(),(1,),(2,3)
```

```
#通过+运算连接两个旧元组，得到一个新元组
```

```
tup1 = tup2 + tup3
```

```
tup1 = tuple("Python!")
```

```
#元组也支持索引和切片运算
```

```
print(f"{tup1[2]}, {tup1[3:6]}")
```

```
#由于元组是不可变对象，元素不可以修改
```

```
#所以该语句报错
```

```
tup1[2] = 'a'
```

```
#代码2.20 元组对象的常用方法
```

```
#tuple()函数可以将参数对象转换成元组
```

```
tup1 = tuple("Beautiful is better than ugly.")
```

```
#统计指定的参数在元组中出现的次数
```

```
print(f" tup1.count('t') = {tup1.count('t')}")
```

```
#查找指定的参数，并返回其在元组中的索引值
```

```
print(f" tup1.index('a') = {tup1.index('a')}")
```

## 为什么有单独的元组和列表数据类型？

虽然列表和元组在许多方面是相似的，但它们的使用方式通常是完全不同的。可以认为元组类似于Pascal记录或C结构；它们是相关数据的小集合，可以是不同类型的数据，可以作为一个组进行操作。例如，笛卡尔坐标适当地表示为两个或三个数字的元组。

另一方面，列表更像其他语言中的数组。它们倾向于持有不同数量的对象，所有对象都具有相同的类型，并且逐个操作。例如，`os.listdir('.')` 返回表示当前目录中的文件的字符串列表。如果向目录中添加了一两个文件，对此输出进行操作的函数通常不会中断。

元组是不可变的，这意味着一旦创建了元组，就不能用新值替换它的任何元素。列表是可变的，这意味着您始终可以更改列表的元素。只有不变元素可以用作字典的key，因此只能将元组和非列表用作key。



# 组合数据类型——字典

- 字典 ( dict ) 是表示映射关系的类型，由**键-值**对构成字典中的元素，是一种可变数据类型
- 字典不是序列类型，其中元素不能使用索引值进行访问，而是通过键访问对应元素，**键必须唯一**，不可变

#代码2.21 字典对象的定义和基本运算

```
dct1 = {} #定义一个空字典
```

#字典元素的键可以是整数、字符串和元组等不可变对象

```
dct1 = {1:"Python", "Tom":"Male", (3,4):"Red"}
```

```
dct1[(3,4)] = "Blue" #字典的元素是可以被修改的
```

```
del dct1[1] #字典的元素可以被删除
```

```
dct1[2] = "Java" #字典中可以随时添加元素
```

```
print(f"After change : {dct1}")
```

#如果一个容器对象中的每个元素都是

#包含2个元素的对象，则可以被转换成字典

```
dct2 = dict([[1,"Python"],["Tom","Male"],[(3,4),"Red"]])
```

#使用zip()函数可以在两个对象之间

#建立元素的对应关系，从而创建字典

```
dct3 = dict(zip(["Tom","Jack","Rose"],["Male","Male","Female"]])
```

# 字典的常用方法



#keys()方法可以返回字典中的所有键

```
print(f"{dct1.keys()}")
```

#values()方法可以返回字典中的所有值

```
print(f"{dct1.values()}")
```

#items()方法可以返回字典中的所有键-值对

```
print(f"{dct1.items()}")
```

#get()方法用来返回以参数1作为键的元素值，如果找不到则返回参数2的内容

```
print(f"{dct1.get('Tom','Unknown')}")
```

```
print(f"{dct1.get('Jerry','Unknown')}")
```

#copy()方法通过拷贝原字典数据创建一个新的字典对象

```
dct2 = dct1.copy()
```

```
print(f"{dct2}")
```

#popitem()方法用于弹出最后被加入字典的元素

```
print(f"{dct2.popitem()}")
```

#pop()方法用于弹出字典中的指定元素，并返回其值

```
print(f"{dct2.pop('Tom')}")
```

#update() 作用是使用参数对象的内容对字典进行更新

```
dct2.update({'Jerry':'Male'})
```

#clear()方法可以将字典中的元素清空

```
print(f"{dct2.clear()}")
```

#fromkeys()用参数1作为键，参数2作为值，创建字典

```
dct3 = dict.fromkeys(["Alice","Mary"],"Female")
```

#setdefault()用于以参数作为键-值对插入元素

```
print(f"{dct3.setdefault('Alice','Male')}")
```



# 字典是如何实现的？

## 字典是如何在CPython中实现的？

CPython的字典实现为可调整大小的哈希表。与B-树相比，这在大多数情况下为查找（目前最常见的操作）提供了更好的性能，并且实现更简单。

字典的工作方式是使用 `hash()` 内置函数计算字典中存储的每个键的hash代码。hash代码根据键和每个进程的种子而变化很大；例如，"Python" 的hash值为-539294296，而"python"(一个按位不同的字符串)的hash值为1142331976。然后，hash代码用于计算内部数组中将存储该值的位置。假设您存储的键都具有不同的hash值，这意味着字典需要恒定的时间 --  $O(1)$ ，用Big-O表示法 -- 来检索一个键。



# 使用list还是dict？

- 场景：学生管理系统，每个学生有学号、姓名、班级、成绩等信息
- 功能1：对于学生成绩进行排序
- 功能2：根据学号修改成绩等信息

- 集合是一种非序列容器对象，其中的每个元素都是唯一的，可变集合（set）表示该集合创建后依然可以对其中元素进行增减或修改，不可变集合（frozenset）表示该集合一旦创建，元素便无法修改

```
#代码2.24 集合对象的定义和基本运算
set1 = {1,2,3,4,5} #定义可变集合对象
set2 = {3,4,5,6,7}
print(f"{3 in set1}") #判断某个元素是否在集合中
print(f"{set1|set2}") #求两个集合的并集
print(f"{set1&set2}") #求两个集合的交集
print(f"{set1-set2}") #求两个集合的差集
print(f"{set2-set1}")

set2 = set1
print(f"{set1}, {set2}")
```

```
#判断集合set1是否为集合set2的真子集
print(f"{set1 < set2}")

#判断集合set1是否为集合set2的子集
print(f"{set1 <= set2}")

set3 = frozenset({1,2,3}) #创建不可变集合
print(f"{set3}")

#不可变集合是无法对元素进行修改的，所以程序报错
set3.add(4)
```

#代码2.25 集合对象的常用方法

```
set1 = {1,2,3,4,5} #定义可变集合对象
```

```
set1.add(6) #向可变集合中添加元素
```

```
print(f"After add: {set1}")
```

#从可变集合中删除元素，如果该元素不存在就报错

```
set1.remove(6)
```

#从可变集合中删除元素

#如果该元素不存在，什么也不做

```
set1.discard(6)
```

```
set2 = {3,4,5,6,7}
```

#求两个集合的交集

```
print(f"{set1.intersection(set2)}")
```

#求两个集合的差集

```
print(f"{set1.difference(set2)}")
```

#用参数中的对象更新可变集合

```
set1.update(set2)
```

#判断set2是否是set1的子集

```
print(f"{set2.issubset(set1)}")
```

#判断set1是否是set2的父集

```
print(f"{set1.issuperset(set2)}")
```

- Python函数与C语言函数基本相似，定义的关键字是**def**，
- Python函数的形式参数可以设置默认值

#代码2.26 从键盘上输入一个正整数，判断其是否为素数

```
def isPrime(n):  
    for i in range(2,n):  
        if n%i==0:  
            return False  
    return True  
num = int(input("请输入一个大于2的正整数："))  
print(f"{num}是素数" if isPrime(num) else f"{num}不是素数")
```

#代码2.27 参数的默认值

```
def fun(a=10, b=20, c=30):  
    return f"{a} + {b} + {c} = {a+b+c}"  
print(fun(15, 25, 35)) #没有使用参数默认值  
print(fun(15, 20)) #参数c使用了默认值30  
print(fun(b=15)) #参数a和参数c使用了默认值  
print(fun()) #所有参数都采用默认值
```



# 引用传递和值传递

- Python中的变量是没有类型的，我们可以把它看做一个(\*void)类型的指针，变量是可以指向任何对象的，而对象才是有类型的。
- Python中的对象有不可变对象（number，string，tuple等）和可变对象之分（list，dict等）。
- 1. 不可变对象作为函数参数，相当于C语言的值传递。
- 2. 可变对象作为函数参数，相当于C语言的引用传递。（因列表是可变数据类型，作为参数传递实则是传递对列表的引用，修改更新列表值后依旧引用不变）
- Python函数可以return多个值

- **lambda**表达式的功能是创建一个非常短小的函数应用，将参数和返回值的关系进行约定

```
g = lambda x: x+1
```

```
g(5)
```

```
#代码2.28 匿名函数lambda表达式的应用
```

```
ages = {"Tom":20, "Jack":19, "Rose":18, "Mary":21}
```

```
#默认的排序方式是按照键的从小到大顺序
```

```
print(f"{sorted(ages) }")
```

```
#在sorted函数中，可以通过指定key参数对应的函数来改变排序的依据
```

```
#以lambda表达式的返回值，即年龄进行排序
```

```
print(f"{sorted(ages, key=lambda x:ages[x]) }")
```

```
1 """
2 A pure Python implementation of the quick sort algorithm
3
4 For doctests run following command:
5 python3 -m doctest -v quick_sort.py
6
7 For manual testing run:
8 python3 quick_sort.py
9 """
10 from __future__ import annotations
11
12
13 def quick_sort(collection: list) -> list:
14     """A pure Python implementation of quick sort algorithm
15
16     :param collection: a mutable collection of comparable items
17     :return: the same collection ordered by ascending
18
19     Examples:
20     >>> quick_sort([0, 5, 3, 2, 2])
21     [0, 2, 2, 3, 5]
22     >>> quick_sort([])
23     []
24     >>> quick_sort([-2, 5, 0, -45])
25     [-45, -2, 0, 5]
26     """
27     if len(collection) < 2:
28         return collection
29     pivot = collection.pop() # Use the last element as the first pivot
30     greater: list[int] = [] # All elements greater than pivot
31     lesser: list[int] = [] # All elements less than or equal to pivot
32     for element in collection:
33         (greater if element > pivot else lesser).append(element)
34     return quick_sort(lesser) + [pivot] + quick_sort(greater)
35
36
37 if __name__ == "__main__":
38     user_input = input("Enter numbers separated by a comma:\n").strip()
39     unsorted = [int(item) for item in user_input.split(",")]
40     print(quick_sort(unsorted))
```

[https://github.com/TheAlgorithms/Python/blob/master/sorts/quick\\_sort.py](https://github.com/TheAlgorithms/Python/blob/master/sorts/quick_sort.py)