第一章 Python 入门导学

1-1 导学

Python 不是一门新兴的语言，上世纪 90 年代初
TIOBE 语言排行 Python 是 第四名 https://www.tiobe.com/tiobe-index/
java C C++ Python C# VB.net PHP Javascript SQL R Ruby Go

Python ┌ Python2
       └ Python3

课程使用 Python3 从三个大方面教学
基础语法：基础变量 ----> 高阶函数
面向对象：讲思维
常见误区，Pythonic，总结经验，原生爬虫

了解语法是编程的先决条件，精通语法是编好程的必要条件

Pythonic ----> 很。Python 简介就是 Python 的特点，Life is simple, i use Python.

Python 能做什么? ----> 爬虫，大数据，测试，Web，AI， 脚本处理

1-2 Python 的特性

语言：Java Python Go…..
框架：以语言为基础构建的一系列基础功能集合

Python 的特性:

1. 简洁，优雅 life is short, I use Python.
2. 跨平台 Windows Linux MacOS
3. 易于学习
4. 极为强大的而丰富的标准库与第三方库
5. Python 是面向对象的语言

1-3 我为什么喜欢 Python

Python 之禅:(之中的 2 句话)

Simple is better than complex.
Now is better than never.
Although never is often better than *right* now.

在 Python 环境中查看 Python 之禅

```
>>>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!
```

1-4 Python 的缺点

慢 ----> (对比与 C,C++,Java)运行效率慢

计算机语言：分两类

- 编译型语言: C, C++
- 解释型语言: Javascript, Python

运行效率 与 开发效率

- 运行效率：开发完代码后 运行代码的速度 (性能)
  - 虽然快 但是 如果代码写的不好也有可能会运行的很慢
- 开发效率：开发代码时 写代码的速度
  - Python 开发效率快

1-5 一个经典误区

编程  Web 编程

可以学习一点 Web 编程 Web 是基础

1-6 Python 能做些什么

(几乎是万能的) 哈?

1. 爬虫
2. 大数据与数据分析(Spark 框架)
3. 自动化运维与自动化测试
4. Web 开发:Flask 框架, Django 框架
5. 机器学习:Tensor Flow
6. 胶水语言:混合其他语言来编程，如: C++，Java
    a. 能够把用其他语言制作的各种模块(尤其是 C/C++)很轻松地联合在一起

1-7 课程内容与特点

- 基础语法　一定要扎实
- Pythonic　很-Python

    代码的复杂程度对比 ---- A B 两个变量交换 值

    Java 实现

    int temp = a;

    a = b;

    b = temp;

    三行代码才实现

    Python 实现:

    a,b = b,a　　一行代码就实现了

- Python 高性能与优化

    写代码不要只满足功能 要选择一种最佳的代码方案 追求高性能

- 数据结构

    也是非常基础的 不同的编程语言都是通用的

1-8 Python 的前景

嗯 就是很有前景 就对了

1-9 课程维护与提问

先 Debug　　微信公众号: 林间有风

QQ 群: 299631895

验证信息:1801281026544986

第二章 Python 环境安装

2-1 下载 Python 环境安装

Python 官网 https://www.python.org/
去 Download 下载 Python2 & Python3

2-2 安装 Python

先安装 Python 3 新建一个文件夹
运行安装文件 勾选 Add Path 3.x to PATH

在安装 Python 2 新建一个文件夹
计算机 右键 properties, advanced system settings
Environment variables, System variables
Path 加一条 Python 2 的安装路径

Rename Python3 文件夹中的 Python.exe 成 Python3.exe

然后在 Shell 中 输入 python 进入 Python 2
输入 python3 进入 Python 3

2-3 IDLE 与第一段 Python 代码

IDLE 分为 Python 2 和 Python 3
用 search bar 找到不同的 IDLE 可以在需要时使用

Python 2 print “hello world”
Python 3 print(“hello world”)

第三章 理解什么是写代码与 Python 的基本类型

3-1 什么是代码, 什么是写代码

代码：是实现世界事务在计算机世界中的映射
写代码：是将现实世界中的事物用计算机语言来描述

3-2 数字:整形与浮点型

Number 数字{int 整数型，float 浮点型，complex 复数，bool 布尔值(True 或 False)}
Python 2 中还有一个类型为 Long 任何精度的整数 只在 Python2 中

int 整数型 正 负都行
float 浮点数 就是带小数点的数
(其他语言中有分 单精度 float 和 双精度 double)
(在 Python3 中没有单双精度的区分！)
(其他语言 在整数型中 有分 short int long)
(在 Python3 中都没有 在 Python 2 中有 long)

type() 函数
type() 函数 如果只有一个参数返回对象的类型

```
type(1)     <class 'int'>
type(1+1.0) <class 'float'>   1+1.0 = 2.0 ----> float
type(1*1.0) <class 'float'>
type(1.1)   <class 'float'>
type(1+1)   <class 'int'>
type(1*1)   <class 'int'>

type(2/2)   <class 'float'>
   2/2 ----> 1.0 ----> float
   2//2 ----> 1 ----> int     // 整除 只留整数部分
```

int / int 如果能整除 结果就是 int 不能整除 结果就是 float
int // int 就算不能整除 也只保留整数部分

3-3 十进制 二进制 八进制 十六进制

十进制 0 1 2 3 4 5 6 7 8 9 10
以 0 开始 满 10 个数字进 1

二进制 0 1 10 11 100 101 110 111 1000
以 0 开始 满 2 个数字进 1

八进制 0 1 2 3 4 5 6 7 10 11 12 13 14
以 0 开始 满 8 个数字进 1

十六进制 0 1 2 3 4 5 6 7 8 9 A B C D E F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20
以 0 开始 满 16 个数字进 1 9 以后的数字以 A B C D E F 来表示

十进制 二进制 八进制 十六进制 中的 10 都代表不容的意思 虽然都是 10 但是意义不同 不是同一个真实的数字

还有其他的进制 时间 60 秒 ----> 1 分钟
60 分钟 ----> 1 小时

3-4 各进制的表示与转换

Python 中表示二进制:

在数字前加上 0b (零 b) 2 进制 --> 10 进制

```
>>> 0b10
2
>>> 0b11
3
```

IDLE 中 用 0b10 自动将 2 进制转到 10 进制 直接回车

Python 中表示八进制:

在数字前加上 0o (零 o) 8 进制 --> 10 进制

```
>>> 0o10
8
>>> 0o13
11
```

IDLE 中 直接会将 8 进制转到 10 进制

Python 中默认为 10 进制 直接输入就好

Python 中表示十六进制:

在数字前加上 0x (零 x)

```
>>> 0x10
16
>>> 0x1F
31
```

IDLE 中直接会将 16 进制 转为 10 进制

各进制转换

1. 其他进制的数 转成 2 进制 bin()

   bin(50) ----> '0b110010' 10 进制 ----> 2 进制

   bin(0o7) ----> '0b111' 8 进制 ----> 2 进制

   bin(0xE) ----> '0b1110' 16 进制 ----> 2 进制

2. 其他进制的数 转成 10 进制 int()

   int(0b111) ----> 7 2 进制 ----> 10 进制

   int(0o777) ----> 511 8 进制 ----> 10 进制

   int(0x1F) ----> 31 16 进制 ----> 10 进制

3. 其他进制的数 转成 8 进制 oct()
   oct(0b111) ----> '0o7'   2 进制 ----> 8 进制
   oct(0x777) ----> '0o3567' 16 进制 ----> 8 进制
   oct(20)   ----> '0o24'   10 进制 ----> 8 进制

4. 其他进制的数 转乘 16 进制 hex()
   hex(888)   ----> '0x378' 10 进制 ----> 16 进制
   hex(0o777) ----> '0x1ff' 8 进制 ----> 16 进制
   hex(0b111) ----> '0x7'   2 进制 ----> 16 进制

3-5 数字: 布尔类型 与 复数

Number 除 int 和 float 外 还有两种数据类型 bool 和 complex

bool 布尔类型 表示真假
complex 表示复数

bool: True 表示为真 T 为大写
    False 表示为假 F 为大写

bool 是 Number 的一个类型 为什么呢?

```
>>> int(True)
1
>>> int(False)
0
```

```
>>> bool(1)
True
>>> bool(0)
False
```

```
>>> bool(2)
True
>>> bool(2.2)
True
```

```
>>> bool(-1.1)
True
>>> bool(0b10)
True
```

bool 只有是非 0 就是 True 如果是 0 或者 None 就是 False
Python 中任何空值 都是 False 非空值 都为 True

```
>>> bool(0)
False
>>> bool(None)
False
```

None 类型
None 类型 表示一个 Null 对象(没有值得对象)。 Python 提供了一个 Null 对象，在程序中表示为 None。
如果一个函数没有显式的返回值，则会返回该对象。None 经常用作可选参数的默认值，以便让函数检测调用者是否为该参数实际传递了值。
None 没有任何属性，在布尔表达式中求值时为 False。

不只是整数类型可以与 bool 互换，字符串也可以。

```
>>> bool('abc')
True
>>> bool(' ')
True
>>> bool('')
False
>>> bool()
False
```

```
>>> bool([1,2,3])
True
>>> bool([])
False
>>> bool({})
False
>>> bool({1,2,3})
True
```

复数 ---- complex
用 j 表示复数 一般不会经常用 以后可以研究一下

3-6 安装 Python

字符串 ---> 非常常用的基本类型
字符串 str 表示文字类型的数据

str{单引号 ''   双引号" " 三引号 }

```
>>> 'Helloworld'
'Helloworld'
>>> "helloworld"
'helloworld'
```

这两个都是可以的 但是区别就是在于
单引号无法表示 字符串中的引号

```
>>> 'let''s'
'lets'
>>> 'let''s go'
'lets go'
```

如果想使用单引号表示 字符串中的引号 则需要用到转义字符 \

```
>>> 'let\'s go'
"let's go"
```

但是双引号可以表示

```
>>> "let's go"
"let's go"
```

无论任何符号 在 Python 中都必须是英文符号

1 与 '1' 不同 因为 1 是 int , 而 '1' 是 str

3-7 多行字符串

三引号 超长的字符串 不适宜阅读
所以 Python 中规定(建议) 每行的宽度为 79 个字符

三引号 可以使三个单引号 或者 三个双引号 要成对出现 可以回车
但是 IDLE 中会在你每一次回车的时候 加上 \n

还是使用三个双引号吧 虽然在 IDLE 中可以使用三个单引号 但是三个单引号代表多行注释

```
>>> """ helloworld
        helloworld
        helloworld"""
' helloworld\nhelloworld\nhelloworld'
```

为什么在 IDLE 中会有\n 出现?
因为在 Python 中 >>> 表示要接收一个输入 无论是可见的 abcd 或者 是不可见的 tab enter 等
IDLE 要把所有的输入都要显示出来，所以会用转义字符 来表示无法显示的字符

```
但是 >>> """helloworld\nhelloworld\nhelloworld"""
        'helloworld\nhelloworld\nhelloworld'
      如果这样收入 同时也会输出一样的字符

      print() 可以吧\n 等转义字符解析并实现
      >>> print("""hello world\nhelloworld\nhelloworld\n""")
      hello world
      hello world
      hello world
```

IDLE 单引号换行

```
>>> 'hello\
    world'
'helloworld'
>>> "hello\
    world"
'helloworld'
```

3-8 转义字符

转义字符 ----> 特殊的字符{表示无法"看到"的字符 换行 等，与语言本身语法有冲突的字符}

转义字符
\n 换行
\` 单引号
\t 横向制表符
etc 还有很多 可以自行百度

\n \r 容易搞混 不是同一个概念 可以找一下 百度 google
\n 换行 \r 回车

print 输出 hello \n world

print('hello \\n world') 这与 let\'s go 同理
这个可以再输出文件夹路径的时候用到
>>> print('C:\\fatherfoldername\\childfoldername')
C:\fatherfoldername\childfoldername

但是要是有太多的 \\ 这样做也是真的很麻烦

所以在路径前面加一个 r 表示 r 后面的字符串不是一个普通字符串 而是一个原始字符串
r 的大小写不影响
这个原始字符串 所见即所得
print(r'C:\fatherfoldername\childfoldername')

注意 这种情况不是普遍适用的
如果字符串里面有单引号就不能用
print(r'let's go') 这种情况就会报错 因为出现了三个单引号
所以换成双引号就没问题了啊
print(r"let's go")

3-9 included_in_3-8 原始字符串

3-10_3-12 字符串运算 一、二、三

合并两个字符串 & 截取字符串

合并 拼接 用 + (加号) * (乘号)

"Hello" + "world" ========> "Helloworld"
"hello" * 3 ========> "hellohellohello"
"hello" * "world" ========> 报错 字符不可以与字符相乘
字符串只能与数字相乘

截取 获取字符串 中的单个字符
"Helloworld"[0] ==> H
"Helloworld"[3] ==> l

只要输入字符的字号就能得到该字符 字号从 0 开始

"H e l l o w o r l d"
"0 1 2 3 4 5 6 7 8 9"

如果中间带空格 空格也算一位
"H e l l o w o r l d"
0 1 2 3 4 5 6 7 8 9 10 这是字号
-11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 这是 n 次的字符

"hello world"[-3] ====> r
字符串后面的方括号[]中 如果是正数 代表字号 从 0 开始
字符串后面的方括号[]中 如果是负数 从该字符串最后向前数 多少个位的字符
-n 就是从后往前数 第 n 个字符

如果要得到 w 在 "hello world" 中
"Hello world"[6] ====> w
"Hello world"[-5] ====> w

截取 获取字符串 中的一组字符 要从截取字符串的起点 并且往后截取几个字符
"hello world"[0:5] ====> hello
"hello world"[0:-1] ====> hello worl 我们发现 d 消失了
这里面的-1 代表了 "步长" 往回数 1 个字符
"hello world"[0:-3] ====> hello wo

Question: 截取 world 的两种不同的方法
"Hello world"[6:11] ====> world 但是奇怪的是 这个字符串没有第 11 个字符 所以无论你写什么数字 只要大于 10 就没问题
"Hello world"[6:] ====> world 可以使用这个样子 系统就明白从第六个字符开始截取 一直截取到字符串结束

"hello python java c# javascript php ruby"
截取 编程语言 去掉 hello
"hello python java c# javascript php ruby"[6:] ====> python java c# javascript php ruby
这样就去掉了 hello

[6:] 从第 6 位向右截取到字符串结束
[:4] 从字符串 0 位向后截取 4 个字符
[:-4] 从字符串 0 位向前截取 4 个字符 字符串第一个字符永远是 0 号字符
[-4:] 从字符串 -4 位向后截取到字符串结尾。 ====> ruby

第四章 Python 中表示 '组' 的概念与定义

4-1 列表的定义

序列 组的概念 就是一个或多个元素组成在一起

列表 也是组的一种 列表 [] [list]

python 中定义列表 [1,2,3,4,5] 这个列表有 5 个元素
完成定义后可以使用 type() 来检测一下
type([1,2,3,4,5]) ======> <class 'list'>

列表中的元素 可以为任何类型的数据 比如 string boolean 或者 不同的数据类型混合在一起也可以。

["Hello","world",1,9,True,False] 多种数据类型混合在一个列表中 作为该列表的不同元素

在列表中 也可以用列表作为元素 这就是 二维列表 或 多维列表
[[1,2],[3,4],[5,6],[7,8],[True,False]]

type([[1,2],[3,4],[5,6],[7,8],[True,False]]) =========> <class 'list'>

二维数组 和 多维数组 在 Python 中叫做嵌套列表。

4-2 列表的基本操作

访问列表中的某一个或多个元素 以 0 位开始

[1,2,3,4,5,6][0] ========> 1 <class 'int'>
[1,2,3,4,5,6][0:2] ========> [1,2] [0:2] 从 0 位开始打印 2 个元素
[1,2,3,4,5,6][-1:] ========> [6]

这跟字符串中的字号相似

[1, 2, 3, 4, 5, 6] 列表
0 1 2 3 4 5 列表的元素序列号
-6 -5 -4 -3 -2 -1 同样也是元素的序列号

列表用 : 英文感叹号访问 多个元素会返回一个列表

更改列表内的元素的值
==========

1. 列表相加 会合成为一个列表
[1,2,3,4,5] + [6,7] =======> [1,2,3,4,5,6,7]

2. 两个列表相乘 没有这种操作 会报错 列表不可以乘列表
列表可以乘数字
[1,2,3,4,5] * 3 =======> [1,2,3,4,5,1,2,3,4,5,1,2,3,4,5]
[列表]*n 会合成为一个列表 并且把该列表重复 n 次 也就是列表中的所有元素乘 n

3. 列表没有减法 没有这种操作 会报错

用嵌套列表的方式来表示世界杯小组赛
A1 B1 C1 D1 E1 F1 G1 H1
A2 B2 C2 D2 E2 F2 G2 H2
A3 B3 C3 D3 E3 F3 G3 H3
A4 B4 C4 D4 E4 F4 G4 H4

[[A1,A2,A3,A4],[B1,B2,B3,B4],[C1,C2,C3,C4],[D1,D2,D3,D4],[E1,E2,E3,E4],[F1,F2,F3,F4],[G1,G2,G3,G4],[H1,H2,H3,H4]]

4-3 元组 tuple

元组用小括号表示 用逗号将元素隔开 元组中的元素可以为不同类型 (元素,元素,元素,元素,元素)

访问元组 元组相加 元祖乘数字 与列表相同

(0,1,2,3,4)[0] ====> 0
(0,1,2,3,4)[0:2] ====> (0, 1)
(0,1,2) + (3,4,5) ====> (0,1,2,3,4,5)
(0,1,2) * 3 ====> (0, 1, 2, 0, 1, 2, 0, 1, 2)

如果元组中只有一个元素 则不会认为是元组类型 而是会认定为元素的类型
type((1)) ====> <class 'int'>

==========
因为在 python 中 () 也可以表示一种数学的运算 如 (1+1)*2
()就叫做运算优先级括号 这就会和元组产生冲突
当有歧义的时候 Python 中规定 当有一个括号并且其中只有一个元素 就当做运算符号来进行计算
==========
但是这在 list 中就不会有这样的顾虑
==========
但如何定义只有一个元素的元组呢？ (1,) 加一个都好加装后面有一个元素
如何定义一个元素都没有的元组呢？ ( ) type(( )) ====> <class 'tuple'>

4-4 序列总结　　序列是有序的

基本类型总结

数字基本类型 (int　float　bool)

str　list　tuple　这三种基本数据类型非常相似 都是序列 str 是由单个字母组成的一串字母 所以叫做字符串

==========

序列有哪些共同特点?

1. 可以在序列后面加 [ ] 进行访问

序列内的每一个元素都会被分配一个序号 通过这个需要对学历进行访问

"String"[2] ====> r

2. 切片　就是序列都能切片

[1,2,3,4,5][0:3]　====>　[1,2,3]

[1,2,3,4,5][-1:]　====>　[5]

'helloworld'[0:8:2]　====>　hloo

3. 序列可以进行 加法 和 乘法

4. 序列中是否存在某个元素　在这里引用一个新的符号　in　 not in　会返回一个 bool 类型

某元素是否在序列中　　3 是否在序列[1,2,3] 中

3 in [1,2,3]　====>　True

某元素是否不在这个序列中

3 not in [1,2,3]　====>　False

in / not in　是逻辑运算符

5. 一个序列中一共有多少个元素　引入一个新的 function　len()

len([1,2,3,4,5,6])　====>　6

len("Hello World")　====>　11

6. 序列中找出最大的元素

max([1,2,3,4,5,6])　====>　6

7. 序列中找出最小的元素

min([1,2,3,4,5,6]) =====> 1

max 和 min 在 str 中会按照在字母表中的顺序输出 空格为最小

ASCII 码 将数字与字母相连接 可以以数字的形式代表字母 可以得出字符串中那个字母的大小。

字母的大小写在 ASCII 中都会有不同的数字表示。

max("hello world") =====> "w"

min("hello world") =====> " "

当你要查看一个数据在 ASCII 中如何表示的时候 使用 ord() function

ord()只接收一个参数 把这个参数转换成 ASCII 码

ord("w") =====> 119

ord("d") =====> 100

ord(" ") =====> 32

如果你想要吧 ASCII 转换成字母的时候使用 chr() 功能就可以了

chr(97) =====> 'a'

4-5 set 集合

除了 元组 tuple () 列表 List[] 以外 另一种基本类型为 集合 set

集合 set 的特性就是 无序

type({1,2,3,4,5,6}) =====> <class 'set'>

集合 set 没有下标索引 所以不支持切片 也不支持用下表索引访问

集合 set 内的元素不允许重复
  {1,1,2,2,3,3,4,4,5,5} =====> {1,2,3,4,5}

集合支持的操作:

  1. 长度判断 len({1,2,3,4}) =====> 4

  2. 是否有其元素 1 in {1,2,3} =====> True
    是否没有其元素 1 not in {1,2,3} =====> False

集合的特殊操作:
  除掉元素
    集合 1 {1,2,3,4,5,6} 集合 2 {3,4,7}

    {1,2,3,4,5,6} - {3,4,7} =====> {1,2,5,6} {1,2,5,6} 叫做差集 这就是两个集合的差集合

  1. 除掉两个集合共有的元素
    集合 1 - 集合 2 大集合 - 小集合

  2. 保留两个集合共有的元素
    集合 1 & 集合 2 {1,2,3,4,5,6} & {3,4,7} =====> {3,4} 这是交集

  3. 合并集合
    {1,2,3,4,5,6} | {3,4,7} =====> {1,2,3,4,5,6,7}
    最后结尾的这个集合 叫做 合集 或 并集

如何定义一个空的集合?
  set()

  验证是否为空 len(set()) = 0

4-6 安装 Python

dict 字典是 Python 的基本数据类型
dict 一般会有两个值 key value
key 就是 关键字 value 就是 相应的数据得值

dict 可以有多个 key 和 多个 value

定义一个 dict:
{key 1: value 1,key 2: value 2,key 3: value 3, ……}

{1:1, 2:2, 3:3}
{'Q':'kill','R':'kiss','w':'kim'}

字典是无序的 所以字典无法使用下标进行访问 所以 dict 通过 key 来访问
{'Q':'kill','R':'kiss','w':'kim'}{'Q'} ====> kill

dict 字典不可以有两个相同的 key
{1:'kill','1':'kiss','E':'kim','R':'business'}
这里面的 1 与 '1' 是不同的 虽然都是 1 但是第一个是数字 int 第二个是字符 string

字典 dict 中 valur 所能选取的 Python 数据类型总结:
Value: 任意 Python 基本数据类型
str int float list set dict

字典中的 value 可以使另外一个字典
type({1:'kill','1':'kiss','E':{1:1},'R':'business'}) ====> dict

key: 必须是不可变的类型 eg: int str
列表不可以 列表是可变的 list 元组也是不可以的 元组也是可变的 tuple

如何定义一个空的字典呢?
一个空的字典 表示就是一个空的大括号 {}
type({}) ====> dict

4-7 思维套图总结 基本数据类型

第五章 变量与运算符

5-1 什么是变量

假设有两个列表 列表 A [1,2,3,4,5,6] 列表 B [1,2,3]

如果 列表 A 乘 3 再加上列表 B 再加上列表 A

[1,2,3,4,5,6] *3 + [1,2,3] + [1,2,3,4,5,6]
这样的输入太麻烦 如果假设 A 是一个拥有 10000 个元素的列表就 GG 了

所以 这里引入变量 变量就好像是名字 用字母 A 代表列表 A B 代表列表 B A 就是列表 A 的名字

定义一个变量 A=[1,2,3,4,5,6] B=[1,2,3] 这里的 = 是赋值符号

print(A) =====> [1,2,3,4,5,6]

所以上面的题目用变量表达就会很简单了

A*3+B+A 就是这么简单 不管你有多少个元素 都囊括在 A 中。。

给变量起名必须要有意义 A 就没有什么意义

action=['eat','run','stay'] action

命名的可读性要强 让别人更好地理解 (不要用拼音 不要用汉式英语 no Chinglish) 所以好好学英语

5-2 变量的命名规则

1. 变量名不能以数字开头

2. 变量名可以由字母、数字、下划线 组成

3. 变量名不可以为系统关键字(Python 系统关键字)
   比如说：and if import 等等
   得到 Python3 中所有的关键字 可以百度搜索 Python 保留关键字
   或者
   在 Python3 的 IDLE 中执行代码

```
>>> import keyword
>>> keyword.kwlist
['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']
```

   type 可以用于变量名 但是 强烈建议不要使用 不要使用！
   因为:

```
>>> type = 1
>>> type(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not callable
```

   在这种情况就会报错 当 type 作为一个变量名 把整型赋给了 type 然后你把 int 当做一个方法来调用 就大错特错了
   所以 type print 等这类的虽然可以用作变量名 但是不要用 never

4. Python 变量名要区分大小写
   a=1 print(A) 这是没有用的
   apple 不等于 Apple

5. 变量本身是没有类型的 字符串、整形、元组都可以复制给变量
   变量是没有类型减值的

情况 1:

```
>>> a=1
>>> b=a
>>> a=3
>>> print(b)
1
```

情况 2:

```
>>> a = [1,2,3,4]
>>> b = a
>>> a[0]
1
>>> type(a[0])
<class 'int'>
```

在这里 a[0] 的情况 与 [1,2,3,4][0]的情况是一样的

当我们对 a 进行一下修改

```
>>> a[0] = '1'
```

这个 1 是字符串的 1

```
>>> print(a)
['1', 2, 3, 4]
>>> print(b)
['1', 2, 3, 4]
```

我们可以看到 a 与 b 同时修改了

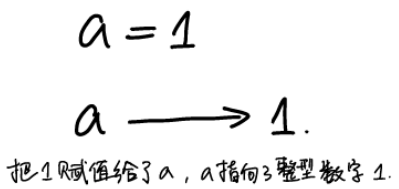
5-3 值类型与引用类型

5-2 中所提到的两种情况 一个是吧 int 类型赋值给了 a 一个是把列表赋值给了 a 两种情况是完全不同的

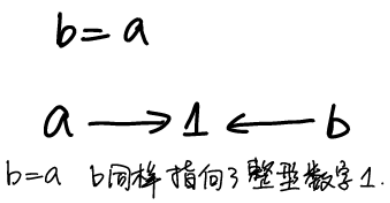
int 类型属于值类型 列表 list 属于 引用类型

什么是值类型呢？ 值类型解释:

当 a=1 把 1 赋值给了 a a 指向了整形数字 1

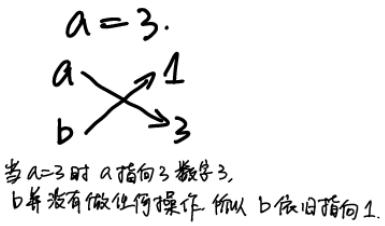


又使得 b = a b 同样也指向了整形数字 1



当 a = 3， a 再一次指向了数字 3

b 没有任何的操作 所以这时候 a=3 b=1



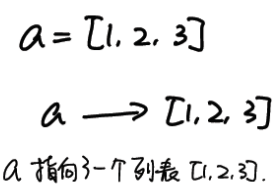
print(a) =====> 3
print(b) =====> 1

值类型的 值是改变不了的 只能重新生成一个新的值。

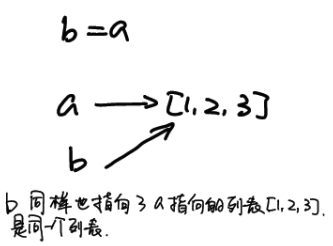
==========

什么是引用类型？ 引用类型解释:

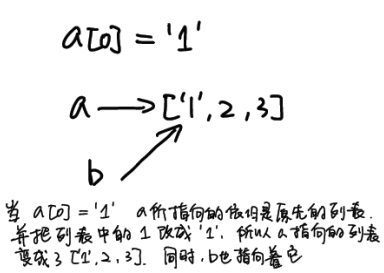
a = [1,2,3]
a 指向了一个列表[1,2,3]



b = a
b 同样指向了那个列表[1,2,3]



a[0] = '1'



引用类型的值可变 值类型的值不可变

这时候 a 并没有指向一个新的列表 而是仍然指向原来的列表 只是把列表更改了 所以 b 让然指向它

总结　int str tuple 是值类型 值类型不可以改变
　　list set dict 是引用类型 引用类型可以改变

eg:

```
>>> a = 'hello'
>>> a = a+'python'
>>> print(a)
hellopython

>>> b = 'hello'
>>> id(b)              id(object) 方法是查看 object 所在内存的地址
4325619504
>>> b = b + 'python'
>>> id(b)
4325628272
```

这两个 id 地址一定是不一样的 因为 b = b + 'python' 得到的是一个新的字符串

都说 str 是一个序列 但是这里说 str 是值类型 所以来验证一下 str 能否被更改

'Python'[0] ==========> 'P'

'Python'[0] = 'A' =========> 这就会报错

5-4 列表的可变 与 元组的不可变

主要讲的是 tuple 和 list 之间的区别

list 列表的特性

```
>>> a=[1,2,3]      定义一个 a 列表
>>> id(a)          把这个列表的内存地址列出来 内存地址一般用 16 进制表示会好一点
57838888           我们发现这个内存的地址为 57838888
                    >>> hex(id(a))
                    '0x3728d28'
>>> a[0] = '1'     当我们改变 a 列表中的第一个元素
>>> print(a)       我们打印 a 确定 a 列表已经被改变了
['1', 2, 3]
>>> id(a)          再次我们打印一下 a 改变后的 id 地址
57838888           我们发现 这是同一个地址  所以列表的更改是在同一个 id 地址上的 而不会建立一个新的.

>>> b=[1,2,3]
>>> b.append(4)     append() 方法用于在列表末尾添加新的对象。
>>> print(b)
[1, 2, 3, 4]
```

tuple 元组的特性

```
>>> a=(1,2,3)                                  设定一个元组 a 元组中有三个元素
>>> a(0) = '1'                                 当我们尝试去更改 a 元组中的第一个元素为 字符串 1 的时候
SyntaxError: can't assign to function call          系统就会报错
>>> a=(1,2,3)                                  设定一个元组 a 同样里面有三个元素
>>> a.append(4)                                  当我们尝试去调用 append 方法在里面增加一个新的元素 4 的时候
Traceback (most recent call last):                  系统报错!
  File "<pyshell#3>", line 1, in <module>
    a.append(4)
AttributeError: 'tuple' object has no attribute 'append'
```

多维数组的访问:

a=(1,2,3,[1,2,4]) 这种元组里面包括列表的形式 叫做 多维元组 现在的 a 是一个 2 维元组

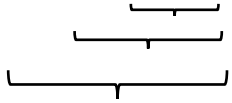
访问 3, a[2] ======> 3

访问 4, a[3] ======> [1,2,4] 我们找到了元组的第四个元素 list [1,2,4]

a[3][2] =====> 4 我们要继续在这个列表中访问第 2 个元素 得到 4

在这里列举一个三维的元组：

a=(1,2,3,[1,2,[a,b,c]])



如果要在这个三维元组中

访问 b: a[3][2][1] ==========> b

多维数组的更改：

还是拿二维元组举例 a=(1,2,3,[1,2,4])

a[3][2] = '3'

修改 2: >>> a=(1,2,3,[1,2,4])

>>> a[1] = 3

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module> 这会报错 因为元组的元素是不能被修改的

a[1] = 3

TypeError: 'tuple' object does not support item assignment

修改 4: >>> a[3][2] = '3'

>>> print(a)

(1, 2, 3, [1, 2, '3']) 只修改 list 就会成功 运行成功

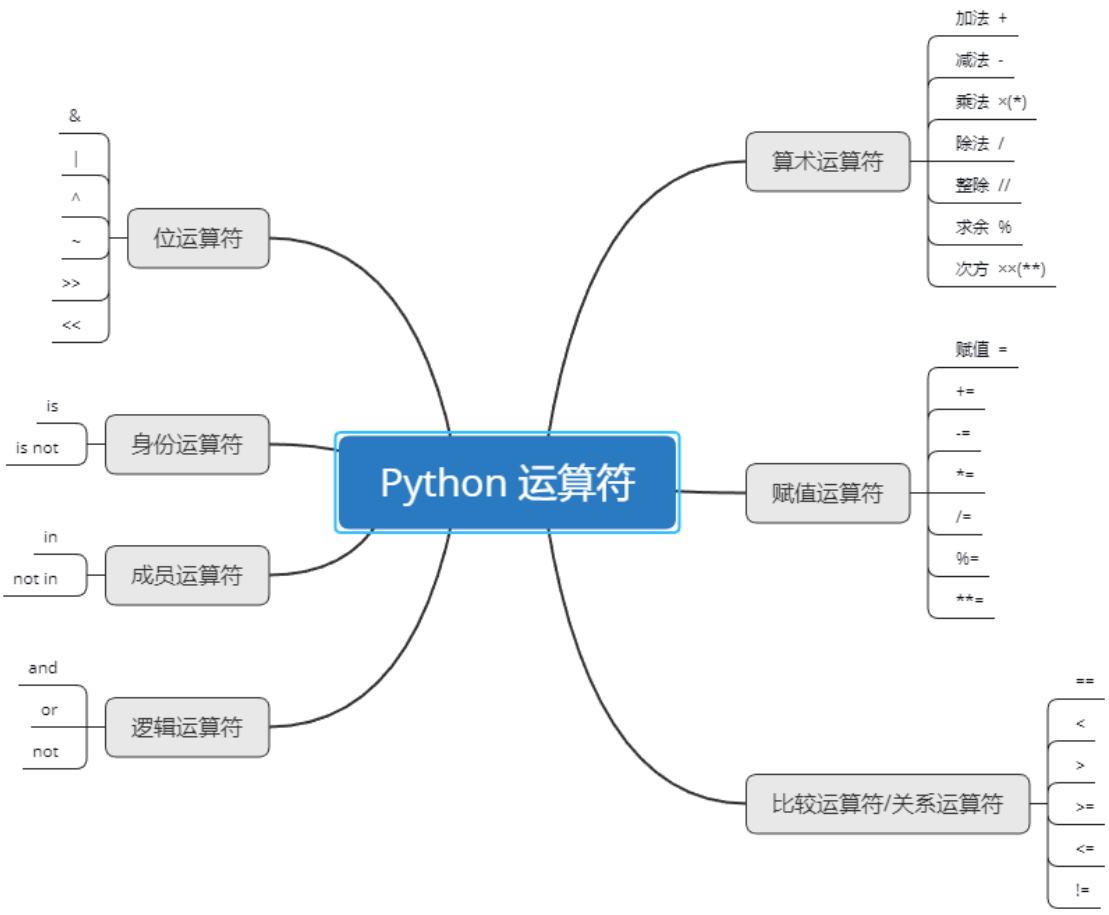
虽然元组 tuple 是不可改变的 但是这是改变数组 list 中的元素 list 中的元素是可以修改的

5-5 运算符号

1+1=2 在这里 加号 + 就是运算符
但是不仅仅是数字 Str 也可以使用运算符进行运算
'hello'+'world'

运算符不仅仅有 加号+ 还有 乘号 ×(*) 减号 - 除号 / 整除号 // 求余符号 %
多次方运算符号 2 次方 (**2) 3 次方 (**3) 4 次方(**4) 以此类推
eg:
100**2 100 的次方 100**3 100 的立方 100**5 100 的 5 次方

运算符有很多的种类 以上所介绍的都算是 算术运算符
算术运算符包括 + - ×(*) / // % ××(**)
加 减 乘 除 整除 求余 次方



5-6 赋值运算符

= 等号 就是赋值运算符

下面的符号都是需要先进性运算再赋值的

```
+=  *=  -=  /=  %=  **=  //=
```

c=1
Python 变量不需要定义 所以上面的代码是 把 1 赋值给变量 c

c=c+1 ======> c+=1
print(c) ======> 2

c++ ++ 自增运算符 ！！ 但是在 Python 中没有自增 自减运算符
c-- -- 自减运算符

==========

加等 +=

```
>>> a=3
>>> b=2
>>> b=a+b
>>> print(b)
5
>>> a=3
>>> b=2
>>> b+=a
>>> print(b)
5
```

减等 -=

```
>>> a=3
>>> b=2
>>> b-=a   # b=b-a  b=2-3 =-1
>>> print(b)
-1
>>> a=3
>>> b=2
>>> b=b-a
>>> print(b)
-1
```

乘等 *=

```
>>> a=3
>>> b=2
>>> b*=a
>>> print(b)
6
>>> a=3
>>> b=2
>>> b=b*a
>>> print(b)
6
```

==========

5-7 比较运算符 关系运算符

比较运算符/关系运算符 是 两个变量之间作比较用的
比较运算符/关系运算符 一共有 5 个操作符:
== != > < >= <=

== 比较两组数据类型是否相等
!= 比较两组数据类型是否不相等
> 大于
< 小于
>= 大于或等于
<= 小于或等于

比较运算符结束后会返回一个 bool 类型的值

```
>>> 1==1    # = 是赋值  == 是比较运算符
True
>>> 1>1
False
>>> 1>=1      # 1 是否大于或等于 1  或运算 有一个满足就好了  因为 1=1 所以 返回 true
True
>>> 1<=1                                    >>> a=1
True                                        >>> b=2
>>> 1!=2      1 是否不等于 2                 >>> a!=b
True                                        True
=========================================================
>>> b=1                 但是如果你要这么写
>>> b+=1                >>> b=1
>>> b>=1                >>> b+=b>=1   #这种情况只会运行 b+=b  而后面的是不会运行的
True                    >>> print(b)
>>> print(b)            2
2
```

b+=b ====> b=b+1 ====> 1=1+2
b>=1 ====> 2>=1 ====> true
这个逻辑是不对的 错的！！！！！

比较运算符 比 赋值运算符 优先 所以要提前算

b>=1 b=1 b>=1 ====> true true=1   应该先算 b>=1 因为 b=1 所以 b>=1 是 true true 可以表示为 1
b+=1 ====> b=b+1 ====> 2   然后再算 b+=1 得出结论 2>=1 ====> true

5-8 不只是数字才能做比较运算

str 也能比较 就是比较 ASCII 码
eg:
```
>>> 'a'>'b'
False
```

两个字母的比较:
```
>>> 'be'>'cd'
False
```

首先会比较 b 与 c， b<c =====> false，就是第一位的两个字母先进行比较
然后就是 false 得出了结论 是不是蒙了啊 总结一下
(b<c 这个是经过比较得出来的结论 但是题目说 b>c 那就不对了啊 就 false 了)

总结：先去第一位的字母相比较 若相同则对比下一个字母 得出结论后再与条件进行对比

列表对比
[1,2,3]<[2,3,4]

列表比较规则类似于字符串比较
首先比较 两个列表的第一个元素 1 与 2 1<2 与题目相符 true

元祖对比
(1,2,3)<(2,3,4)

规则同上 首先比较两个远足的第一个元素 1=1 相等 然后比较两个元组的第二个元素 2<3 与题目相符 所以 ===> true

5-9 逻辑运算符

主要操作 bool 类型 返回结果也是 bool 类型

逻辑运算符有三个: and(且/与) or(或) not(非)

```
============================================================
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

and 总结: 但凡有假 那就会是假的

```
============================================================
>>> True or False
True
>>> True or True
True
>>> False or True
True
>>> False or False
False
```

or 总结: 只要有 True 存在 那就是为真 只有两个假的情况下才会为假

```
============================================================
>>> not True
False
>>> not False
True
```

not 总结: 不是真的就是假的了 不是假的就是真的了

```
>>> not not False
False
============================================================
```

```
>>> 1 and 1
1
>>> not 1
False
>>> not 0
True
>>> not 2
False
>>> not 3.14
False
>>> not 0.1
False
```

对于 int 和 float 类型， 0 被认定为 False 非 0 被认定是 True

==========

```
>>> 'a' and 'b'
'b'
>>> 'a' or 'b'
'a'
>>> not 'a'
False
>>> not 'b'
False
>>> not ' ' # 这不是一个空字符串 因为这里有一个空格
False
>>> not '' #这才是真正的空字符串
True
>>> not 'a'
False
```

对于 str 字符串 若是空字符串 被认为 False 否则 认定为 True

==========

```
>>> not []
True
>>> not [1,2,3]
False
```

对于 列表 list 空列表被认为 False 否则为 True

==========

tuple 元组，dict 字典 与列表 list 相同。

```
>>> [1] or []
[1]
>>> [] or [1]
[1]
>>> not []
True
>>> not [1]
False
>>> not {}
True
>>> not {1:2}
False
```

对于 tuple 元组，dict 字典 空的元组(tuple) 空的字典(dict) 被认定为 False
非空的元组(tuple) 和 非空的字典(dict) 被认定为 True
==========

当我们知道 int 1 和 2 都代表 True 的时候 我们运行如下代码

```
>>> 1 and 2
2
```

我们可以看到返回的结果为 2 而不是 1 为什么呢?

因为计算机先读取了 1 在读取了 2 进行对比 都是 True 所以按照就近原则 返回了 2
同理

```
>>> 2 and 1
1
```

这证明了返回结果中有按照就近原则

```
>>> 1 or 0
1
```

计算机 读取了 1 (1 为 True) 读取了 or (只要有一个 True 就返回 True) 就不会再继续往下读取了 并且返回 1

```
>>> 1 or 2
1
```

这个理由同上

5-10 成员运算符

成员运算符 有两个: in not in

主要特点 判断一个元素是否在另外一组元素中 并且返回一个 bool 类型的值

in 元素是否在另一组元素中
a in [1,2,3,4,5] =====> Error 因为 a 没有定义

a=1 a in [1,2,3,4,5] =====> True a 等于 1， 1 是不是在列表[1,2,3,4,5]中 返回结果为真
b=6 b in [1,2,3,4,5] =====> False b 等于 6， 6 是不是在列表[1,2,3,4,5]中 返回结果为假
==========

not in 元素是否不在另一组元素中

a=1 a not in [1,2,3,4,5] =====> False a 等于 1， 1 是不是不在列表[1,2,3,4,5]中
1 在列表中 返回结果为假
b=6 b not in [1,2,3,4,5] =====> True b 等于 6， 6 是不是不在列表[1,2,3,4,5]中
6 不在列表中 返回结果为真
==========

以上举例了 成员运算符在 list 中的使用 下面介绍在 string tuple set 和 dict 中的使用

str b='h' b in 'hello' =======> True 字符 h 在 字符 hello 中 对不对

tuple b='h' b not in (1,2,3,4,5) =======> True 字符 h 在元组(1,2,3,4,5)中 对不对

set b='h' b not in {1,2,3,4,5} =======> True 字符 h 在集合{1,2,3,4,5}中 对不对

dict b='a' b in {'c':1} =====> False
b='1' b in {'c':1} =====> False
b='c' b in {'c':1} =====> True

字典 dict 的成员运算是针对 key:Value 的 Key 并不是 Value

5-11 身份运算符

(这里需要理解对象的定义)(现在只是一个简单的了解)

身份运算符: is　is not　并且返回一个 bool 值 用于比较两个对象的存储单元(内存地址)

is 是判断 2 个表示符是不是引用子一个对象

x is y 类似于 id(x) == id(y)
如果引用的是同一对象 则返回 True 否则(两个表示符不是同一对象) 则返回 False

x is not y 类似于 id(x) != id(y)
如果引用的不是同一对象 则返回 True 否则(两个表示符引用的是同一对象) 则返回 False

```
>>> a=1
>>> b=2
>>> a is b
False
```

```
>>> a=1
>>> b=1
>>> a is b
True
```

```
>>> a='hello'
>>> b='world'
>>> a is b
False
```

```
>>> c='hello'
>>> a is c
True
```

```
>>> a=1
>>> b=1.0
>>> a==b
True          # 这是单纯比较 a 和 b 的值 就是 1==1 True
>>> a is b
False         # 这是比较 a 和 b 的内存地址 所以会返回 False
```

```
>>> id(a)
271112368
>>> id(b)
55816320
```

```
>>> a={1,2,3}
>>> b={2,1,3}
>>> a==b
True
>>> a is b
False
```

集合 set 是无序的 所以只比较值
a==b ==> True
id(a) != id(b)
a is b ==> False

```
>>> c=(1,2,3)
>>> d=(2,1,3)
>>> c==d
False
>>> c is d
False
```

c 与 d 是元组 tuple 是有序的 要按照顺序来 所以
c != d ==> False
id(c) != id(d) ==> False

5-12 如何判断变量的值, 身份 与 类型

值 Value 身份 id 类型 type

type 类型的判断
>>> a=1
>>> type(a)==int
True

如果这样做就太麻烦了

函数(Function) isinstance(变量名称要判断的对象,类型名称甚至可以为元组) 返回一个 bool 类型

isinstance(a,int) ====> True

如果为元组:
isinstance(a,(int,str,float))
a 是不是 int str float 中的一种? 同样也会返回 一个 bool 类型

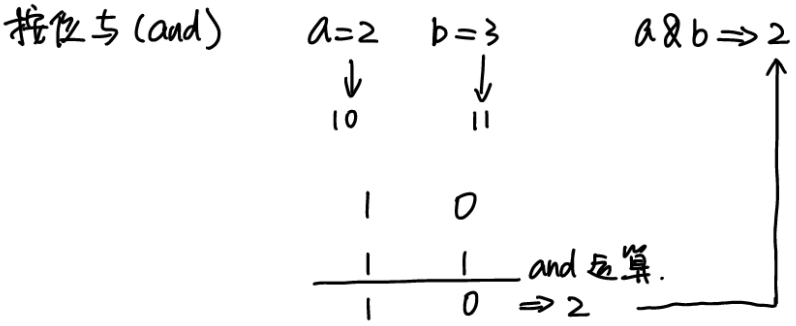
>>> isinstance(a,(int,str,float))
True

5-13 位运算符

&	|	^	~	<<	>>
按位与	按位或	按位异或	按位取反	左移动	右移动

所有的位运算符 都是把数字当做二进制数字进行运算
如果是其他进制的数 会先转成二进制再运算

按位与 (and) a=2 ==> 10
b=3 ==> 11
a&b ==> 2



按位或 (or) a=2 ==> 10
b=3 ==> 11
a|b ==> 3