

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
```

在这种情况下就会报错 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

当我们对 a 进行一下修改

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'>
```

```
>>> a[0] = '1'
这个 1 是字符串的 1
>>> print(a)
['1', 2, 3, 4]
>>> print(b)
['1', 2, 3, 4]
```

在这里 a[0] 的情况 与 [1,2,3,4][0]的情况是一样的

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 一个是把 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

$a = 1$
 $a \longrightarrow 1.$
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

$b = a$
 $a \longrightarrow 1 \longleftarrow b$
 $b = a$ b 同样指向了整形数字 1.

当 a = 3, a 再一次指向了数字 3

$a = 3.$
 $a \longrightarrow 3$
 $b \longrightarrow 1$
当 a=3 时 a 指向了数字 3.
b 并没有做任何操作 所以 b 依旧指向 1.

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]

$a = [1, 2, 3]$
 $a \longrightarrow [1, 2, 3]$
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]

$b = a$
 $a \longrightarrow [1, 2, 3]$
 $b \longrightarrow [1, 2, 3]$
b 同样也指向了 a 指向的列表 [1, 2, 3].
是同一个列表.

a[0] = '1'

$a[0] = '1'$
 $a \longrightarrow ['1', 2, 3]$
 $b \longrightarrow ['1', 2, 3]$
当 a[0] = '1' a 所指向的依旧是原先的列表.
并把列表中的 1 改成 '1'. 所以 a 指向的列表
变成了 ['1', 2, 3]. 同时, b 也指向着它

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

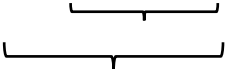
```
>>> a=(1,2,3)          设定一个元组 a 元组中有三个元素
>>> a(0) = '1'         当我们尝试去更改 a 元组中的第一个元素为 字符
串 1 的时候
SyntaxError: can't assign to function call    系统就会报错
>>> a=(1,2,3)          设定一个元组 a 同样里面有三个元素
>>> a.append(4)         当我们尝试去调用 append 方法在里面增加一
个新的元素 4 的时候
Traceback (most recent call last):           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>

被修改的

a[1] = 3

TypeError: 'tuple' object does not support item assignment

这会报错 因为元组的元素是不能

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

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'

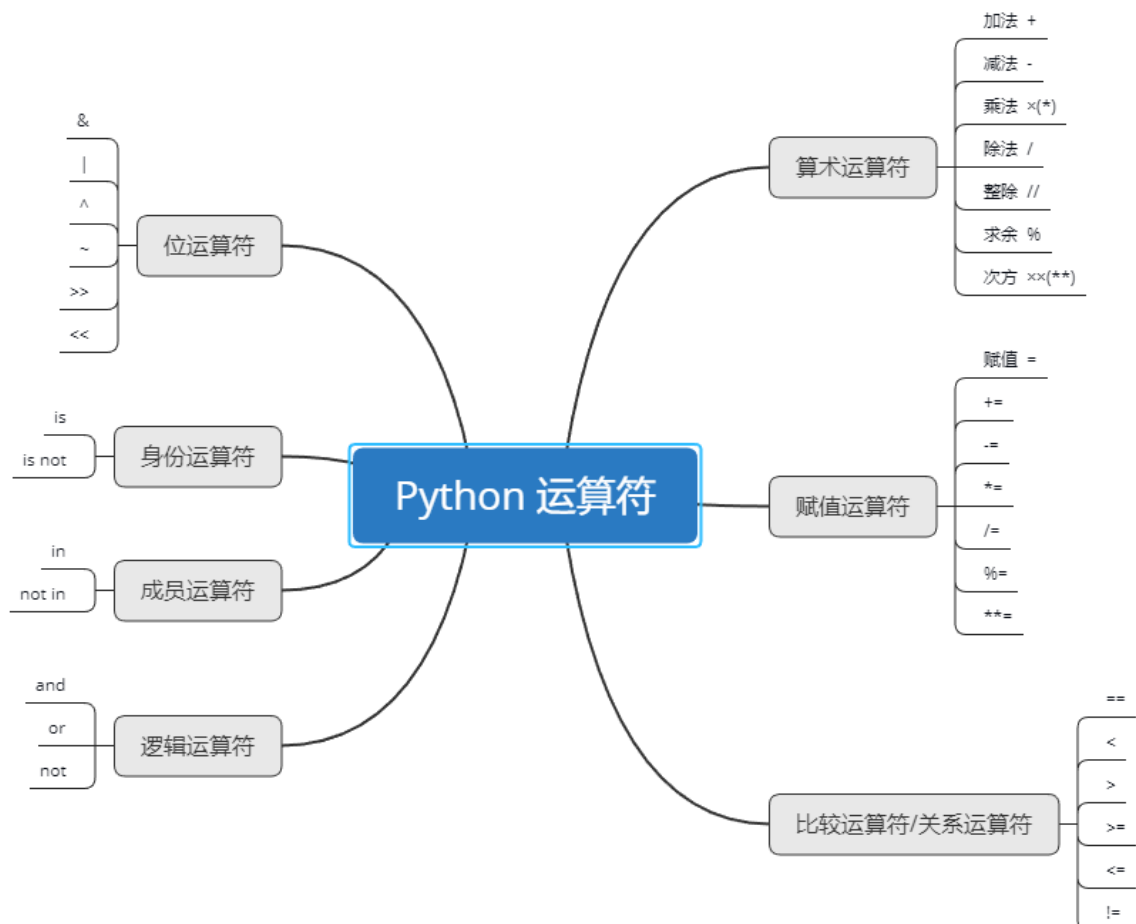
运算符不仅仅有 加号+ 还有 乘号 \times (*) 减号 - 除号 / 整除号 // 求余符号 %
多次方运算符 2 次方 (**2) 3 次方 (**3) 4 次方 (**4) 以此类推

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 + - \times (*) / // % $\times\times$ (**)
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 +=          减等 -=          乘等 *=

>>> a=3          >>> a=3          >>> a=3
>>> b=2          >>> b=2          >>> b=2
>>> b=a+b        >>> b-=a    # b=b-a b=2-3=-1 >>> b*=a
>>> print(b)     >>> print(b)     >>> print(b)
5                -1                6
>>> a=3          >>> a=3          >>> a=3
>>> b=2          >>> b=2          >>> b=2
>>> b+=a         >>> b=b-a         >>> b=b*a
>>> print(b)     >>> print(b)     >>> print(b)
5                -1                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
```

```
True
```

```
>>> 1!=2    1 是否不等于 2
```

```
True
```

```
>>> a=1
```

```
>>> b=2
```

```
>>> a!=b
```

```
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`