# 赋值语句

a=10
a存储的是10的首地址

a=1+1
先运算，再赋值

| 运算 | 说明 |
| --- | --- |
| a='abc' | 基本赋值 |
| a,b='A','B' | 元组赋值 |
| [a,b]=['A','B'] | 列表赋值 |
| a,b='AB' | 序列赋值 |
| a,*b='ABCDE' | python3的序列解包 |
| a=b=c='A' | 多目标赋值 |
| a+=1 | 增强赋值语句 |

- 基本赋值

1. 变量中不存储数据，只存储数据的首地址
   变量中存储了数据的首地址，称之为：引用
   引用>变量
   l=[1,2,3]
2. 优先计算等号右边的数据
3. python3.6引入数字的新写法：
   数字增强
   a=100_000_000
   说明： _并不参与编译，最终的结果和去掉下滑线一致
   作用：仅仅增加可读性，不影响任何代码的执行

```
a='abc'
a='A'+'B'    # a:'AB'
a=100000000
数字增强
a=100_000_000
```

- 序列赋值

1. 和等号右边的序列的元素的位置有关，一一对应
2. 等号两边的元素要一致
   等号右边的数据，不能过多，也不能过少
3. 序列赋值等号右边的数据必须是可迭代对象

```
a,b=1,2
print(a,b)
# a,b=1,2,3
# print(a,b)
```

```
# a,b=1,
# print(a,b)
# a,b=1.5
# print(a,b)
# a,b,c,d=range(4) # 0 1 2 3
# print(a,b,c,d)
# a,b,c=range(4)
# print(a,b,c)
# a,b,c=range(2)
# print(a,b,c)

# a,b,c,d='hehe'
# print(a,b,c,d)

s='hehe'
(a,b),c=s[:2],s[2:] # (a,b),c='he','he'
print(a,b,c)
# 序列切割问题
l=[1,2,3,4]
while l:
    a,l=l[0],l[1:]
    print(a,l)


l=[1,2,3,4,5]
# l[l.index(5)],l[0]=l[0],l[l.index(5)]
l[0],l[l.index(5)]=l[l.index(5)],l[0]
# a=l[0]  b=l[l.index(5)]   l[0]=b
```

```
# [5,2,3,4,5]
# l[l.index(5)]=1
# [1,2,3,4,5]

print(l)

a=10
b=20
a,b=b,a
```

- Python3的序列解包

```
a , *b=1 , 2 , 3 , 4
a:1
b:[2,3,4]
1. 序列打包
        变量前添加一个星号，将剩余的所有数据打包成一个列表
        优先让其他变量先赋值

        a,*b,b=1,2,3,4 # 序列顺序： 从左到右
        # 先加载所有的变量  然后再进行逐个赋值
        print(a,b) # 加载之后的顺序
        a,b,*b=1,2,3,4
        print(a,b)  # *b
2. 序列解包
```

```
        使用时，依然使用   *变量   表示：将多个元素拆解为多个参数
        a,*b=1,2,3,4
        print(*b)
```

```
# a,*b=1,2,3,4,5
# print(a,b)
# a,*b=1,2
# print(a,b)
# a,*b=(1,)  # *b可以不赋值,b可以是空列表
# print(a,b)
#
#
# a=1,
# print(a)
#
# (a,b)=(1,2)
# print(a)

# a,*b=1,2,3,4
# print(a,b)
#
# a,*b=1,
# print(a,b)
#
# a,*b,c,d=1,2,3,4,5,6,7  # *b  前:从左向右赋值后：从右向左赋值
# print(a,b,c,d)
```

```python
# a,b,*c,d,e,f=1,2,3,4,5
# print(a,b,c,d,e,f)
#
# a,*b,*c=1,2,3,4  # 不可以同时多个序列打包，星号只能有一个
# print(a,b,c)

# a,*b,b=1,2,3,4 # 序列顺序： 从左到右
# # 先加载所有的变量  然后再进行逐个赋值
# print(a,b) # 加载之后的顺序
#
# a,b,*b=1,2,3,4
# print(a,b)  # *b

# a,*b=1,2,3,4
# print(*b,type(*b))
# print(b)
def ceshi(x):pass

a,*b=1,2,'a',4
ceshi(*b)

# 序列分割问题
l=[1,2,3,4]
while l:
    a,*l=l
    print(a,l)
```

- 多目标赋值

```
a=b=c=10
```

- 增强赋值语句

```
a+=1    # a=a+1
1. 增强赋值语句：
            优点：效率高
            缺点：不安全
2. 如何选取：
            需要提高效率：增强赋值
            需要提高安全性：普通赋值
```

---