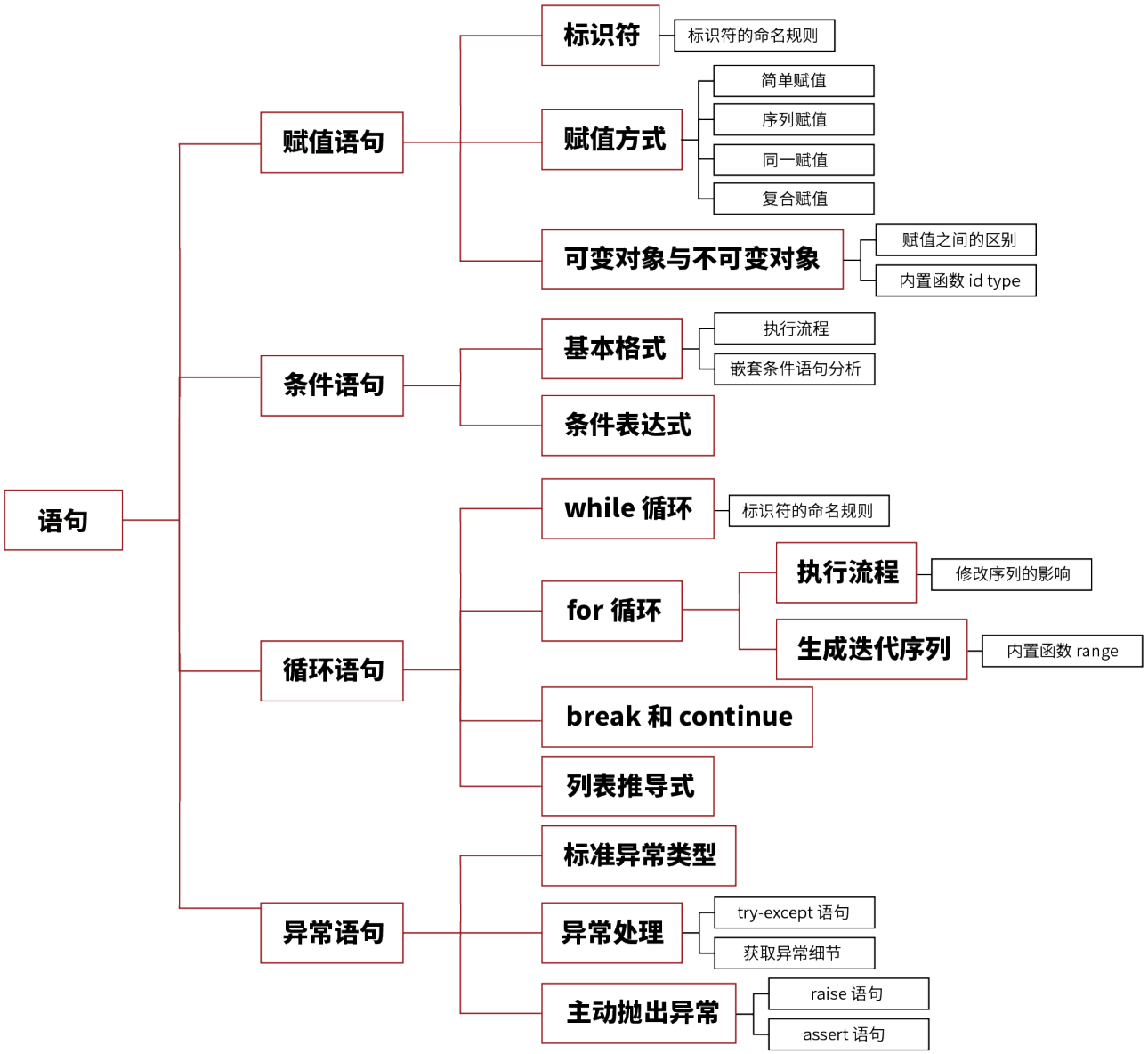
第 3 讲　语句

知识脉络

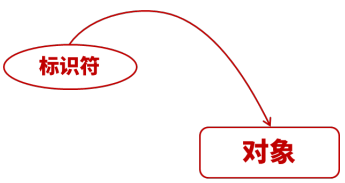


一　赋值语句

1. 标识符

· 用来指代特定对象的名称，将标识符与对象绑定后，后续代码出现该标识符时将用该对象替换

· 标识符的名称可以随意取，但是有以下限制：



① 由字母、数字、下划线组成

② 不能以数字开头，区分大小写

③ 关键字不能被用作标识符（见下表，标红的是本课程没学过的）

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

2. 赋值语句

① 单赋值

```
name = expression    # 计算出 expression 的值后将标识符 name 与其进行绑定（建立上图中的箭头）
```

· **赋值语句不是表达式，因此不存在返回值**（注意不是 None！）

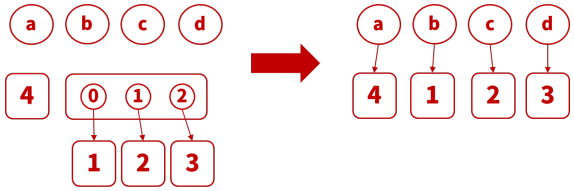
② 序列赋值

```
a, b, … = exp1, exp2, …    # 先计算出右侧所有表达式的值，然后按下面的规则分配
```

· 如果两边标识符和表达式的个数相等，按顺序赋值

· 如果标识符的个数多于表达式的个数，且表达式的结果中有**复合类型**，会尝试将其进行**解包**（拆成单个元素）

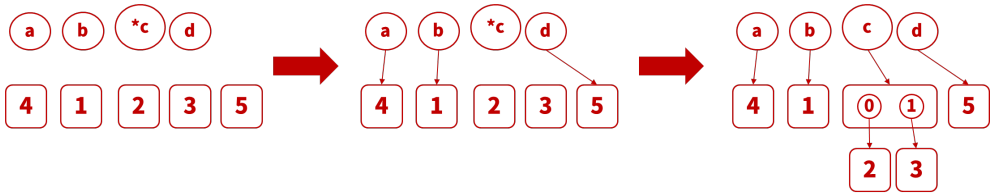
若解包后个数相等，则按顺序赋值，否则报错



· 注意：字典被拆分赋值时，是把键赋值给标识符

· 若有标识符前加了*号（只允许一个标识符加*号），则解包后值的个数**可以多于**标识符个数

按照顺序分配好其它标识符后，将剩下的值（**不管是一个还是多个**）打包成列表，与*标识符绑定



应用：

```
a, b = b, a    # 两个标识符指向的对象可以用一行语句进行互换
```

③ 多个标识符赋同一个值

```
a = b = c = value    # 结果：a、b、c 三个标识符都会指向 value
```

④ 复合赋值

```
a += value
```

```
a -= value
```

```
a *= value
```

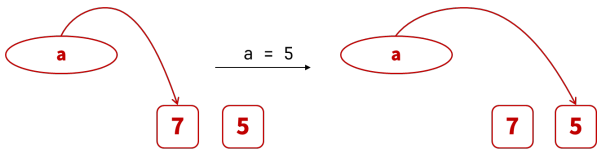
```
a /= value
```

· 首先计算出 value 的值，然后将其与标识符 a 指向的值进行对应的运算，结果赋值给 a

3. 可变对象与不可变对象

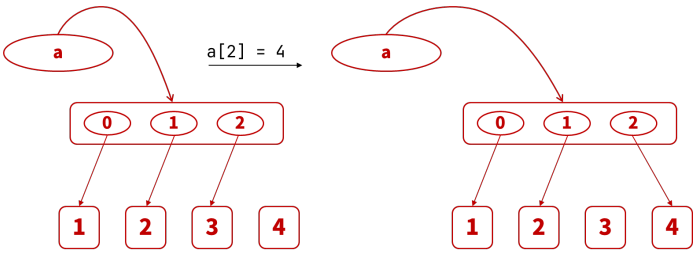
前两讲介绍的数据类型中，列表、集合、字典为可变对象，其余为不可变对象。

· 当标识符指向**不可变对象**时，修改不可变对象的结果是标识符指向新对象



· **可变对象**实质上并不存储元素对象本身，而是元素对象的地址（引用）

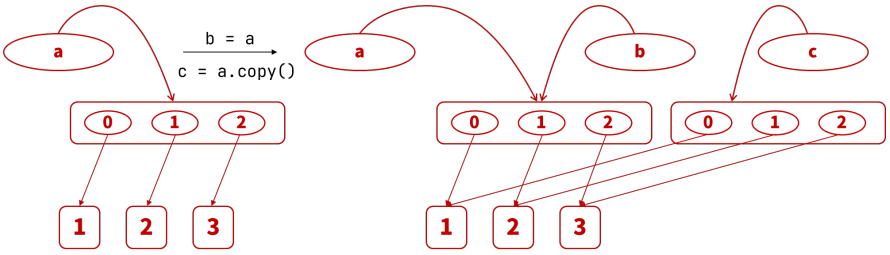
因此标识符指向可变对象时，修改可变对象，标识符依然指向原对象，但对象内部发生变化



- 若有多个标识符指向可变对象，不管通过哪一个标识符修改可变对象，都会使得可变对象发生变化

  而对于不可变对象，其它标识符指向的依然是原对象

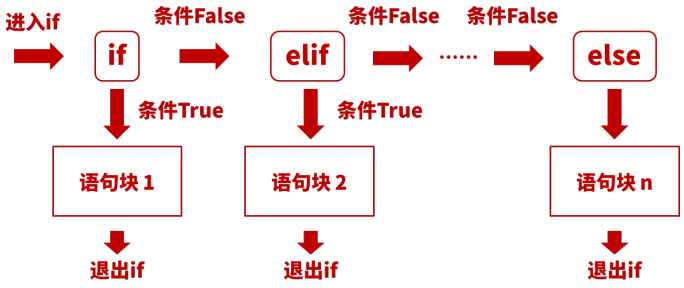
  所以列表的 copy 方法产生了内容与原列表一致的**新列表**，与其它标识符绑定后井水不犯河水



二　条件语句

1. 基本格式

```
if condition1:
    语句块 1
elif condition2:
    语句块 2
…
else:
    语句块 n
```



- 程序将依次检测各个 `condition`，如果 `condition` 为 `true`，将执行对应的语句块，然后退出

  否则继续检查下一个 `condition`，若到达最后的 `else`，直接执行 `else` 的语句块

  `if` 的数量必须是 1 且要放在开头，`elif` 无限制但必须夹在中间，`else` 若有必须放在最后

2. 嵌套条件语句分析

```
if code == 'R':
    if count <20:          # 由于 python 强制缩进的规则，嵌套 if 比 C 语言的好分析多了
    elif:
    else:                  # elif、else 与同缩进的上方最近的 if 匹配
```

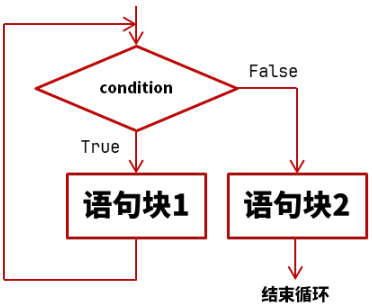
3. 条件表达式

```
exp1 if condition else exp2
```

- 计算表达式 `condition`，若为 `True`，计算并返回表达式 `exp1` 的值；否则计算并返回 `exp2` 的值

三　循环语句

1.while 循环




```
while condition:
    语句块 1
```

```
while condition:
    语句块 1
else:
    语句块 2
```

- 重复执行语句块 1，直到不满足 condition，然后执行语句块 2
- 注意：**循环体内必须要有能改变 condition 结果的代码或者 break，否则将陷入无限循环**

2.for 循环

① 执行流程

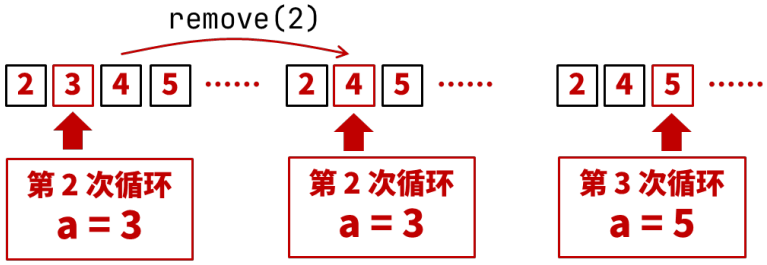
```
for a in seq:
    语句块 1
```

```
for a in seq:
    语句块 1
else:
    语句块 2
```

- 依次将序列 seq 中的元素**赋值**给 a（迭代变量）并执行语句块 1（称为**迭代**），完毕后执行语句块 2

- seq 可以是任一复合类型（注意：**字典作为迭代序列时，迭代变量遍历的是字典的键**）
  - 若 seq 是**字典**，**迭代变量遍历的是字典的键**（若要遍历字典的值，需先通过方法获得序列）
  - 若 seq 是**列表**，**在 for 循环中修改列表**导致**后方元素前移至当前的迭代位置**：

    迭代变量依然指向原元素，且进入**下一轮循环**后指向下一个位置的元素，**直接跳过这个前移的元素**



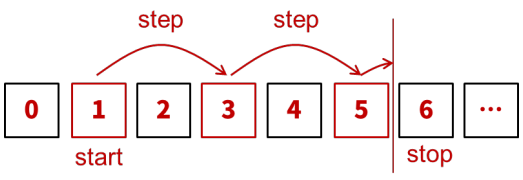
② 生成迭代序列：range 函数

```
range(stop)
```

```
range(start, stop)
```

```
range(start, stop, step)
```

- 返回如下图所示的整数序列（可以看成是对该序列的切片），start 默认为 0，step 默认为 1



3. break 和 continue

① `break` `# 在循环内使用，执行到这句时，跳出整个循环，因此要和条件语句一起使用`

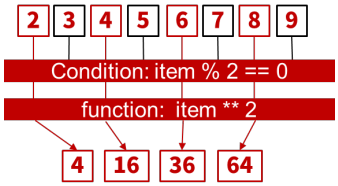
注意：**即使有 else 语句块，break 跳出循环时也不会去执行 else 的语句块**

② `continue` `# 在循环内使用，执行到这句时，跳过当次循环剩余内容，进入下一轮循环`

4. 列表推导式

```
[func(item) for item in seq]
```

```
[func(item) for item in seq if condition]
```



· 从原序列 seq 中筛选出使 condition 为 True 的元素

然后将这些元素代入表达式 func 中，将它们的返回值组成列表，可认为是对列表中的每个元素都做相同的处理

四　异常处理语句

1. 标准异常类型

Python 执行代码时，如果发现错误，会中止程序并抛出异常说明情况

本课程可能会考的标准异常类型如下

异常名称	描述	一般何时会出现
ZeroDivisionError	除(或取模)零 (所有数据类型)	执行除法运算 0 作除数时
IndexError	序列中没有此索引(index)	序列[出界的 index]
OSError	操作系统错误	文件不存在
KeyError	映射中没有这个键	字典[不存在的 key]
TypeError	对类型无效的操作	函数(类型不符合要求的参数)
ValueError	传入无效的参数	函数(值不符合要求的参数)

2. try-except 语句

通过该语句可以让 python 发现错误时不中止程序，转而执行你设定的代码

① 基本流程

```
try:
    语句块 1          👈 首先执行这一部分的语句
except 异常类型:
    语句块 2          👈 如果执行语句块 1 期间发生了对应类型的异常，立即跳转到对应语句块并执行
……                    👈 可以指定多种异常类型及其对应的代码块
except:
    语句块 N+2        👈 如果发生的异常不包括在上面的异常中，则跳转到这里
else:
    语句块 N+3        👈 如果没有发生异常，正常语句块执行完毕后跳转到这里（可省略）
finally:
    语句块 N+4        👈 最后不管是否发生异常，都正常语句块执行完毕后跳转到这里（可省略）
```

② **获取异常细节**

```
except 异常类型 as e:
```

· e 为异常对象，可以通过 str 函数转化为描述异常细节的字符串

3. **主动抛出异常**

主动抛出异常的语句通常回合 try—except 语句结合，以应对多样的异常情况

① raise 语句

```
raise 异常类型(message)    # 主动抛出指定异常类型，message 为异常细节说明字符串
```

② assert 语句

```
assert 判断表达式，描述表达式（可省略）  # 如果判断表达式的值为假，抛出 AssertionError
```

· 该 AssertionError 描述内容为描述表达式的值

典型例题

例 1 下列 Python 语言标识符正确的是

A.2you　　B. my-name　　C._for　　D.abc*234　　E.else

解 A 以数字开头，错；B 和 D 包含非法字符-和*，错；E 为关键字，错；C 正确

例 2 判断：若 z 已赋值，x=(y=z+1)语句是错误语句

解 赋值语句不是表达式，不存在返回值，不能被赋给另一个标识符，因此是错误语句，说法正确

例 3 下面程序的输出是_____.

```
t=1
t,a=2,t+2
print(a)
```

解 等号右侧代入 t=1 后为 2,3，因此分别赋给 t 和 a：t 为 2，a 为 3，因此输出 3

例 4 下面程序输入 18 和 1 时，输出分别是_____。

```
num = int(input())
a = num - 1
while a > 1:
    if num % a == 0:
        print("不是素数")
        break
    a = a - 1
else:
    print("不是素数也不是非素数")
```

解 这个程序首先通过 input 函数输入了字符 18，通过 int 函数转换为数字 18　num 为 18

因此 a=17，因为 17>1，进入循环

观察循环内的语句，发现影响 a 的值的语句只有 a=a-1，也就是说，每次循环 a 都会自减 1

这意味着 a 将随着循环经历 17，16，15……直到 1（不再大于 1）

然后看循环内的其它语句：如果 num（18）能被 a 整除，就会输出不是素数，然后退出循环

a 能够遍历的这些数字中，9 能够被 18 整除，因此输出“不是素数”，直接退出循环

再看输入 1，此时 a 为 0，一开始就不满足循环的条件，因此跳过循环体，直奔 else

也就是输出“不是素数也不是非素数”

如果你能直接看出来这是一个判断素数的程序，用数学常识就能得到答案

例 5 下面程序的输出是_____。

```
l = [i + j for i in range(1, 6) for j in range(1, 6)]
print(sum(l))
```

解 列表生成式中包含了两个 for，说明有嵌套循环，其中靠右的 for 为外循环

外循环为 j 从 1~5，内循环为 i 从 1~5，i j 相互独立，因此列表一共有 5^2=25 个元素

我们用一张表格来更加清楚地表示

i/j	1	2	3	4	5
1	2	3	4	5	6
2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9
5	6	7	8	9	10

sum(l)意味着要将所有元素相加，计算可以得到结果为 150，因此输出的是 150

例 6 下面程序的输出是_____。

```
l = [[(i, j) for i in range(1, 6)] for j in range(1, 6)]
print(l[2][1][0])
```

解 这个程序首先通过一个嵌套的列表生成式生成了列表 l，然后用 print 函数输出了该列表的

第 3 个元素中的第 2 个元素中的第 1 个元素

首先要得到 l 具体生成了啥。l 内部的[]是一个列表生成式，被嵌套在外部的列表生成式中，

因此 l 由 5 个列表元素组成。

以第一个列表元素为例，此时 j=1，因此该列表为[(i, 1) for i in range(1, 6)]

也就是[(1, 1), (2, 1), (3, 1), (4, 1), (5, 1)]，同理可得其它几个列表元素

· 我们需要的第三个元素 l[2]为[(1, 3), (2, 3), (3, 3), (4, 3), (5, 3)]

这个列表的第 2 个元素 l[2][1]为元组(2, 3)

这个元组的第 1 个元素 l[2][1][0]就是 2，因此程序输出的是 2

例 7 下面程序的输出是_____。

```
mat = [[i*3+j+1 for j in range(3)] for i in range(5)]
mattrans = [[row[col] for row in mat] for col in range(3)]
print(mattrans[1][3])
```

解 这个程序首先通过一个嵌套的列表生成式生成了嵌套列表 mat：

由 for i in range(5)，i 将遍历 0 1 2 3 4，依次将 i 代入，得到

mat = [[0*3+j+1 for j in range(3)], …, [4*3+j+1 for j in range(3)]]

由 for j in range(3)，j 将遍历 0 1 2，依次将 j 代入，得到

mat = [[1, 2, 3], [4, 5, 6], …, [13, 14, 15]]

然后基于 mat 用列表生成式生成了列表 mattrans：

由 for col in range(3)，col 将遍历 0 1 2，依次将 col 代入，得到

[[row[0] for row in mat], [row[1] for row in mat], [row[2] for row in mat]]

由 for row in mat，row 将遍历[1, 2, 3], [4, 5, 6], …, [13, 14, 15]:

```
[row[0] for row in mat] = [[1, 2, 3][0], [4, 5, 6][0], …, [13, 14, 15][0]]
                        = [1, 4, 7, 10, 11]
```

同理 mattrans = [[1, 4, 7, 10, 13], [2, 5, 8, 11, 14], [3, 6, 9, 12, 15]]

可以看出，实质上 mat 代表一个 5×3 的矩阵，而 mattrans 是矩阵 mat 的转置

最后输出 mattrans 第 2 个元素中的第 4 个元素，也就是[2, 5, 8, 11, 14]的第 4 个元素

因此最后输出的是 11

例 8 下面程序输入−3 时的输出是_____。

```
def area(r):
    assert r >= 0, "参数错误,半径小于 0"
    return 3.14159 * r * r

r = float(input())
try:
    print(area(r))
except AssertionError as msg:
    print(msg)
```

思路 → 执行 try：r=-3，然后执行函数 area(-3)

→ （函数 area）执行 assert 语句：判断表达式 -3 >= 0 结果为假

→ 抛出 AssertionError，同时将第二个表达式的值"参数错误,半径小于 0"作为异常描述

→ 执行对应的该异常 except AssertionError as msg，msg 接收异常描述"参数错误,半径小于 0"

→ 执行 print(msg)，输出参数错误,半径小于 0

例 9 下面程序的输出是_____。

```
s=0
a, b = 1, 2
if a>0:
    s=s+1
elif b>0:
    s=s+1
print(s)
```

解　a 为 1，b 为 2，if 的条件 a>0 成立，因此执行 s=s+1，此时 s 变成 1

然后直接退出 if，来到 print(s)，因此程序的输出为 1

例 10 下面程序的输出是_____。

```
try:
    x = float("abc123")
    print("数据类型转换完成")
except IOError:
    print("This code caused an IOError")
except ValueError:
    print("This code caused an ValueError")
```

思路　→ 执行 try 中的代码：float 函数的参数"abc123"不符合这个函数要求（含有字母）

→ 触发 ValueError 异常

→ 有 except ValueError，因此跳转执行这一段

→ print("This code caused an ValueError")，输出 `This code caused an ValueError`