東南大學 SOUTHEAST UNIVERSITY 人工智能学院

Python编程

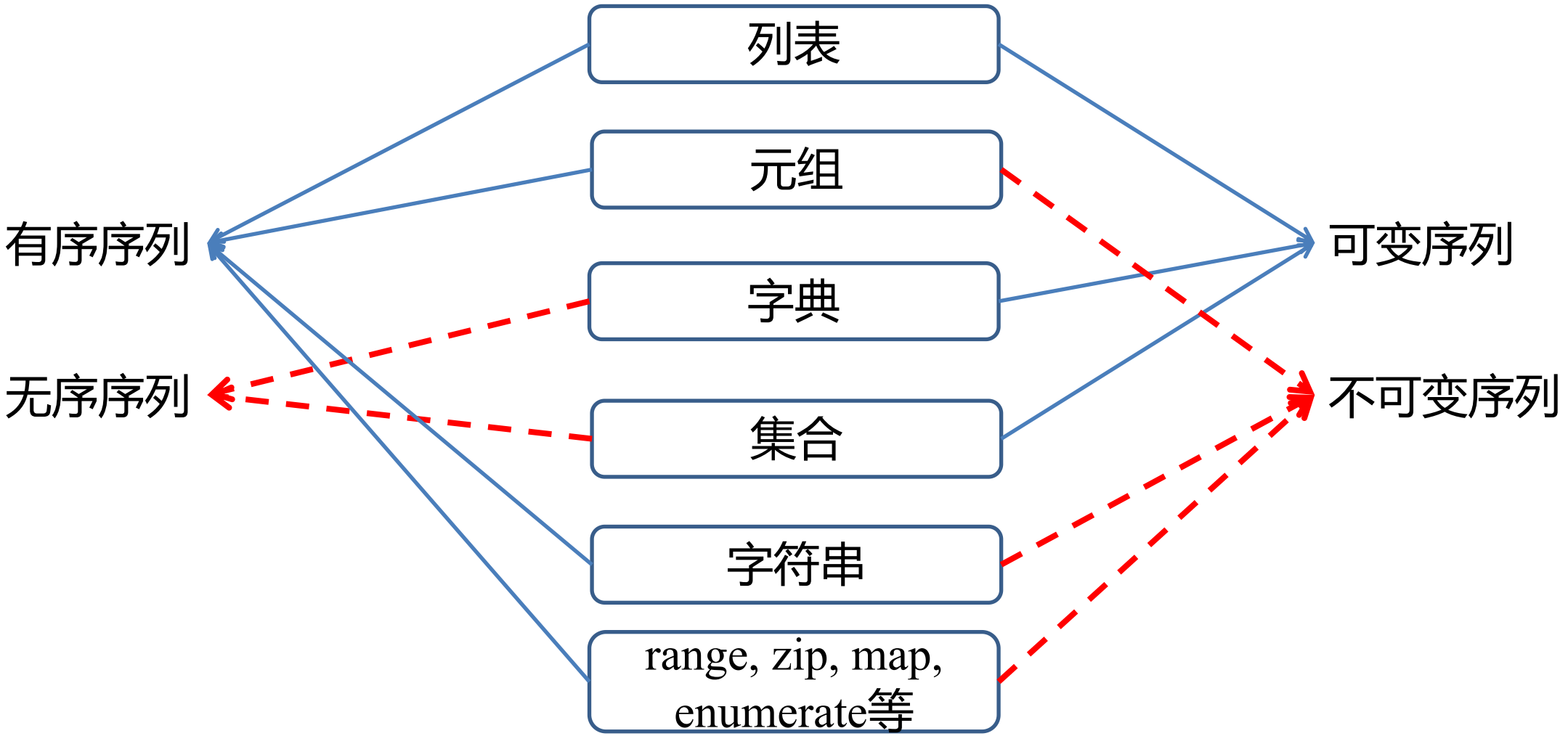
授课教师：王洪松

邮箱：hongsongwang@seu.edu.cn

Python编程复习

- 使用缩进来定义代码块，使用换行符来表示语句的结束
- 变量在赋值时自动声明，不需事先声明变量名及其类型
- Python变量名对英文字母的大小写敏感
- 万物皆对象
- is与==的使用区别，is比较内存地址，==比较内容
- 切片：使用2个冒号分割的3个数字来完成：list[start: stop: step]
- 赋值、浅复制与深复制
- 可变数据类型包括：列表、字典、集合、自定义对象
- 不可变数据类型包括：整数、浮点数、复数、字符串、元组等
- zip()和*zip()，zip()将对象中对应的元素打包成一个个元组

Python序列结构



Python基础知识

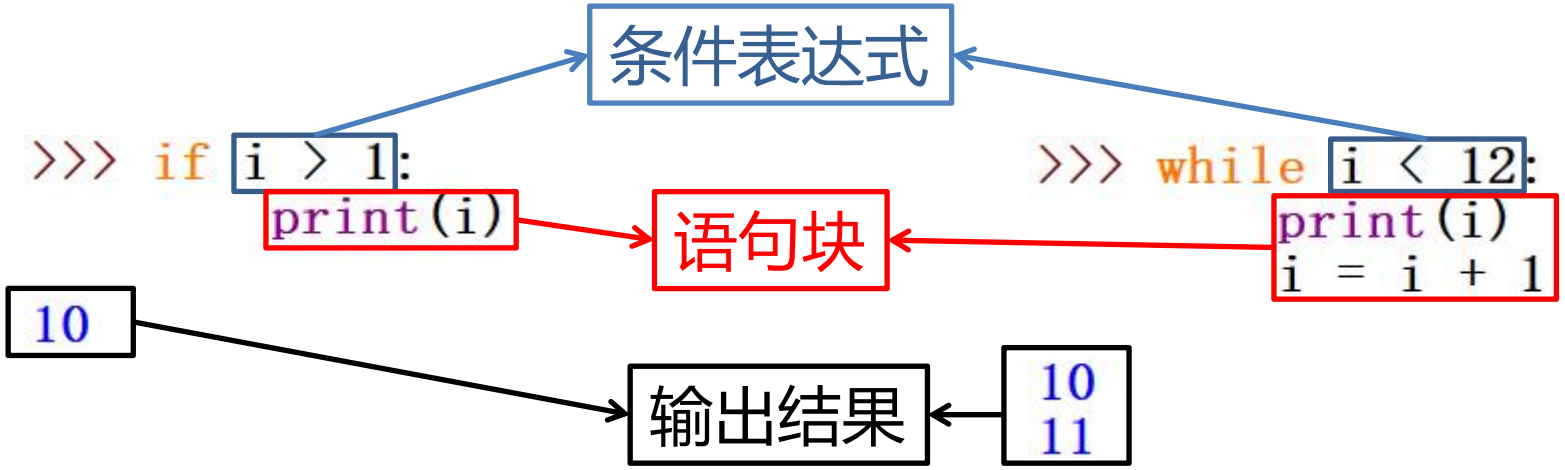
- 函数：内置函数

abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Python选择与循环结构

Python选择与循环结构

- Python程序设计中，有3种常见结构
  - 顺序
    - 自上向下，顺序执行代码
  - 分支(选择)
    - 根据条件判断，决定执行代码的分支
  - 循环
    - 根据条件判断，让特定代码重复执行



Python选择与循环结构

- 条件表达式一般由操作数和运算符组成
  - 运算符
    - 算术运算符
      - +、 -、 *、 /、 //、 %、 **
    - 关系运算符
      - >、 <、 ==、 <=、 >=、 !=（可连续使用）
    - 测试运算符
      - in、 not in、 is、 is not
    - 逻辑运算符
      - and、 or、 not
    - 位运算符
      - ~、 &、 |、 ^、 <<、 >>
    - 矩阵运算符
      - @

Python选择与循环结构

- 在选择和循环结构中，条件表达式的值只要不是False、0（或0.0、0j等）、空值None、空列表、空元组、空集合、空字典、空字符串、空range对象或其他空迭代对象，Python解释器均认为与True等价
- Python语言的合法表达式可以作为条件表达式，包括含有函数调用的表达式

Python选择与循环结构

- 条件表达式结果非0非空，即视为真

```
>>> if 666:
        print(666)

666
>>> if -666:
        print(666)

666
```

```
>>> a = [0]
>>> if a:
        print(666)

666
>>> a = 0
>>> if a:
        print(666)

>>> a = []
>>> if a:
        print(666)

>>>
```

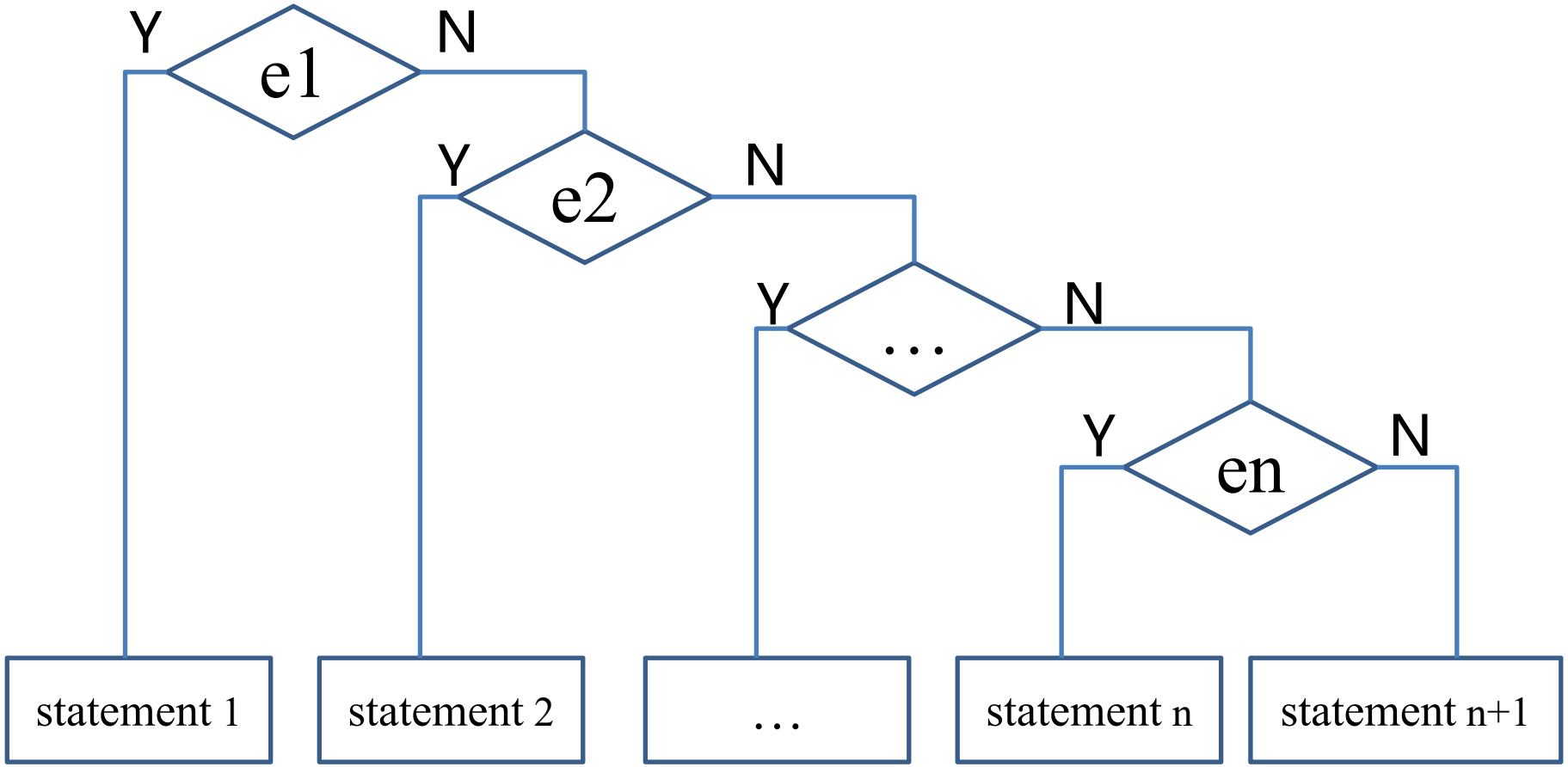
Python选择与循环结构

- 条件表达式
  - 逻辑运算符and和or具有惰性求值的特点，即不管后面的正不正确，先执行前面的判断
  - 对于表达式“表达式1 and 表达式2”
    - 如果“表达式1”值为False，整个表达式的值即为False，“表达式2”的值不会被计算
    - 根据不同条件失败的概率来设计表达式，可以大幅度提高程序运行效率
  - 条件表达式中，不允许使用赋值运算符“=”

```
>>> if a = 3:
SyntaxError: invalid syntax
```

Python选择结构

- if选择结构



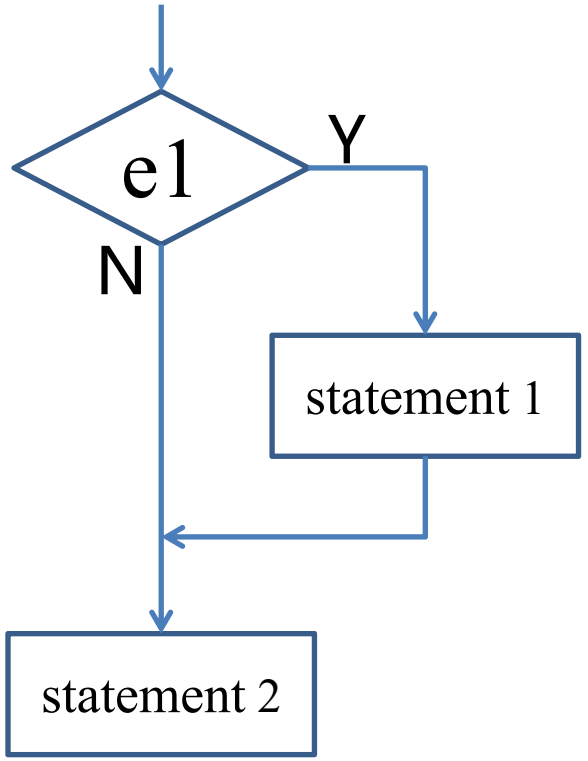
Python选择结构

- 单分支: 只有1个if判断

```
>>> if '表达式':
        '语句块'
```

```
>>> x = input("请输入两个数字:")
请输入两个数字:10 15
>>> a, b = map(int, x.split())
>>> if a < b:
        a, b = b, a

>>> print(a, b)
15 10
```

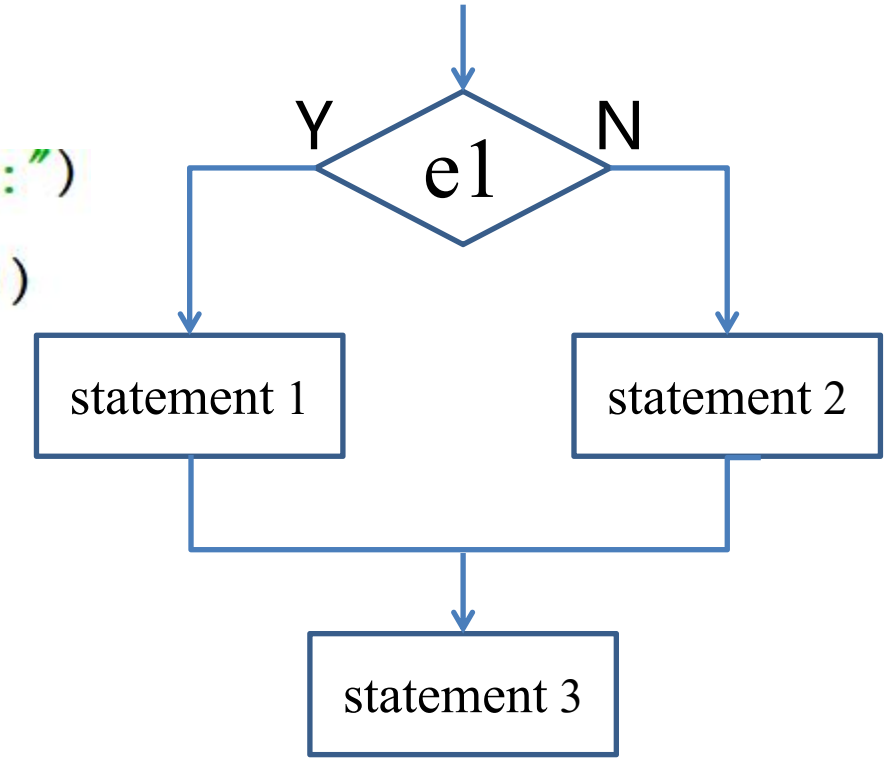


Python选择结构

- 双分支: 一个if判断，一个else

```
>>> x = input("请输入两个数字:")
请输入两个数字:10 15
>>> a, b = map(int, x.split())
>>> if a > b:
        a, b = b, a
else:
        a = b

>>> print(a, b)
15 15
```



Python选择结构

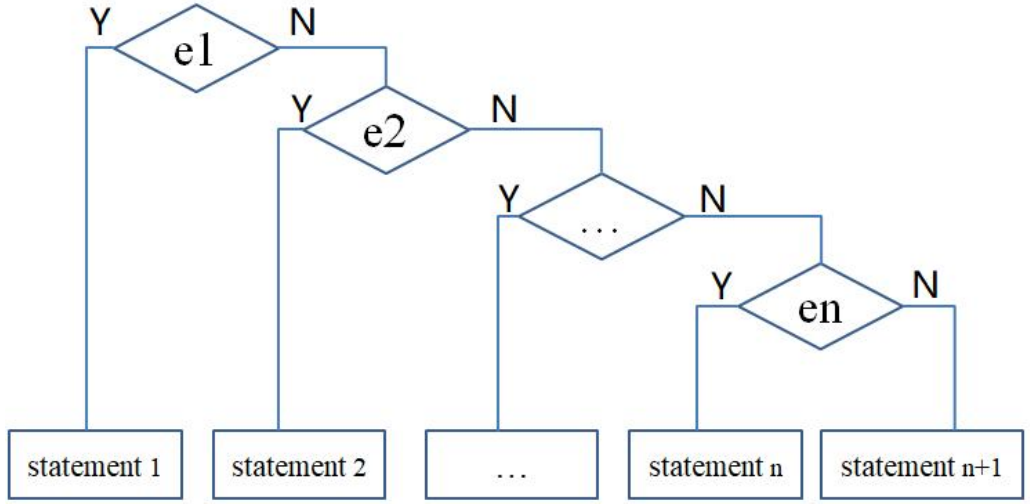
- Python还支持如下形式的双分支选择结构
- value1 if expression else value2
  - 当条件表达式expression 的值为True时，上面表达式的值为value1，否则为value2。
  - 这种结构的表达式同样具有惰性求值的特点

```
>>> a = 5
>>> print(6) if a<6 else print(9)
6
>>> b = 5 if a>9 else 9
>>> b
9
```

Python选择结构

- 多分支：if…elif…else…

```
>>> if '表达式1':
        '语句块1'
elif '表达式2':
        '语句块2'
elif '表达式3':
        '语句块3'
elif '表达式4':
        '语句块4'
else:
        '语句块5'
```



Python选择结构

- 利用多分支结构将百分制分数转换为等级制

```
>>> score = 61
>>> if score > 100:
        print('Wrong data!')
elif score >= 90:
        print('A')
elif score >= 80:
        print('B')
elif score >= 70:
        print('C')
elif score >= 60:
        print('D')
elif score >= 0:
        print('E')
else:
        print('Wrong data!')

D
```

Python选择结构

- 利用选择结构的嵌套将百分制分数转换为等级制

```
>>> degree = 'DCBAAE'    #等级制
>>> score = 61
>>> if score > 100 or score < 0:
        print('Wrong data!')
else:
        index = (score - 60) // 10
        if index >= 0:
                print(degree[index])
        else:
                print(degree[-1])

D
```

- 缩进必须要正确且前后呼应

Python循环结构

- Python提供了两种基本的循环结构语句
  - while语句、for语句
- while循环一般用于循环次数难以提前确定的情况，也可以用于循环次数确定的情况
- for循环一般用于循环次数可以提前确定的情况，尤其是用于枚举序列或迭代对象中的元素
- 相同或不同的循环结构之间都可以互相嵌套，实现更为复杂的逻辑

Python循环结构

- for循环和while循环都可以带else语句

```
while 条件表达式:
    循环体
[else:          #如果循环是因为break结束，则不执行else中语句块
    代码块]

for 取值 in 序列或迭代对象:
    循环体
[else:
    代码块]
```

Python循环结构

- break语句在while循环和for循环中都可以使用，一般放在if选择结构中，一旦break语句被执行，将使得整个循环提前结束
- continue语句的作用是终止当前循环，并忽略continue之后的语句，然后回到循环的顶端，提前进入下一次循环。

Python循环结构

- 例：计算小于100的最大素数

```
# coding=utf-8
for n in range(100, 1, -1):
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            break
    else: #注意缩进
        print(n)
        break
```

n取值100, 99, 98, 97

n取值97

- 在for循环完整完成后才执行else；如果中途从break跳出，则连else一起跳出

Python循环结构

```
# coding=utf-8
for n in range(100, 1, -1):
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            break
    else: #注意缩进
        print(n)
        # break
```

- 删除最后一个break，输出100以内所有素数

Python循环结构

- 警惕continue可能带来的问题：

```
i = 0
while i < 10:
    if i % 2 == 0:
        continue
    print(i)
    i += 1
```

死循环

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

输出1 3 5 7 9

Python循环结构：推导式

- 分析如下四个例子

```
nums = [1, 2, 3]
squares = [n**2 for n in nums]
print(list(squares))
print(sum(squares))
```

```
[1, 4, 9]
14
```

```
nums = [1, 2, 3]
squares = (n**2 for n in nums)
print(list(squares))
print(sum(squares))
```

```
[1, 4, 9]
0
```

```
nums = [1, 2, 3]
squares = [n**2 for n in nums]
print(sum(squares))
print(tuple(squares))
```

```
14
(1, 4, 9)
```

```
nums = [1, 2, 3]
squares = (n**2 for n in nums)
print(sum(squares))
print(tuple(squares))
```

```
14
()
```

Python循环结构：推导式

- 推导式：从一个数据序列构建另一个新的数据序列的结构体
  - 列表推导式、元组推导式、字典推导式、集合推导式

- 列表推导式（列表解析，list comprehension）的语法形式为

```
[expression for expr1 in seq1 if conditoin1
                for expr2 in seq2 if conditoin2
                ...
                for exprN in seqN if conditoinN]
```

Python循环结构：推导式

```
a_List = [x * x for x in range(10)]
```

输出　a_List=[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

等价于：

```
a_List = []
for x in range(10):
    a_List.append(x*x)
```

等价于：

```
a_List = list(map(lambda x: x*x, range(10)))
print(a_List)
```

Python循环结构：推导式

- 元组推导式是用 () 圆括号，列表推导式用的是 []
- 列表推导式生成一个新的列表，但使用元组推导式生成的结果并不是一个元组，而是一个生成器对象
- 可使用 for 循环遍历生成器对象，可以获得各个元素
- 使用 __next__() 方法遍历生成器对象，也可以获得各个元素

- 字典推导式和集合推导式用的是大括号{}

```
tuple1 = (1,1,2,3,4,5,6,6)
setnew = {x**2 for x in tuple1 if x%2==0}
print(setnew)
```

输出结果：

```
{16, 4, 36}
```

Python循环结构：推导式

- 无论是使用 for 循环遍历生成器对象，还是使用 __next__() 方法遍历生成器对象，遍历后原生成器对象将不复存在，这就是遍历后转换原生成器对象却得到空元组的原因

```
a = (x for x in range(1,10))
for i in a:
    print(i,end=' ')
print(tuple(a))
```

输出结果：
1 2 3 4 5 6 7 8 9 ()

```
a = (x for x in range(3))
print(a.__next__())
print(a.__next__())
print(a.__next__())
a = tuple(a)
print("转换后的元组：",a)
```

输出结果：
0
1
2
转换后的元组： ()

Python循环结构：推导式

- 字典推导式和集合推导式用的是大括号{}
- 字典推导式生成的是字典，集合推导式生成的是集合。

```
list1 = ['Jack','Tom']  #将列表中各字符串值为键，各字符串的长度为值
newdict = {key:len(key) for key in list1}
print(newdict)
```

输出结果：{'Jack': 4, 'Tom': 3}

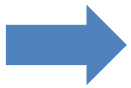
```
tuple1 = (1,1,2,3,4,5,6,6)
setnew = {x**2 for x in tuple1 if x%2==0}
print(setnew)
```

输出结果：
{16, 4, 36}

Python循环结构：扩展知识

- 循环结构的安全问题
  - Python for循环中的索引变量会泄露围合(enclosing)的函数作用域

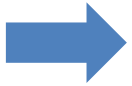
```
for i in [1, 2, 3]:
        pass
print(i)
```



3

```
lst = []
for i in range(4):
    lst.append(lambda: i)

print([f() for f in lst])
```



[3,3,3,3]

Python循环结构：扩展知识

- 循环结构的安全问题
  - for循环将变量赋值到目标列表中
  - 当循环结束时，赋值列表中的变量不会被删除
  - 如果序列是空的，它们将不会被赋值给所有的循环
  - Python中最内层的可能作用域是一个函数体，不是一个for循环体，不是一个with代码块

```
for i in [1, 2, 3]:
    pass
print(i)  # 3
```

```
for i in []:
    pass
print(i)   # NameError异常
```

Python循环结构：扩展知识

- 再看一个例子，输出是多少?

```
nums = [1, 2, 3]
squares = (n**2 for n in nums)
print(4 in squares)
print(9 in squares)
```

True
True

```
nums = [1, 2, 3]
squares = (n**2 for n in nums)
print( 9 in squares)
print( 4 in squares)
```

True
False

```
nums = [1, 2, 3]
squares = (n**2 for n in nums)
print( 9 in squares and 4 in squares)
print( 4 in squares)
```

False
False

Python循环结构：扩展知识

- while循环
  - 本质就是一个条件语句，自定义条件，当条件满足的时候，不断执行while代码块
- for循环
  - Python中的for循环有一些新奇之处

```
int[] integers = {1, 2, 3};
for (int j = 0; j<integers.length; j++) {
        int i = integers[j];
        System.out.println(i);
}
```

传统for循环

```
numbers = [1, 2, 3]
for i in numbers:
        print(i)
```

Python中的for循环

Python循环结构：扩展知识

- for循环
  - 没有索引变量
  - 没有变量的初始化、边界检查和值的增长

- for循环是否在底层使用了索引?
  - 不使用索引，使用迭代器

```
numbers = [1, 2, 3]
i = 0
while i < len(numbers):
        print(numbers[i])
        i += 1
```

Python循环结构：扩展知识

- for循环的底层实现逻辑

```
def funky_for_loop(iterable, action_to_do):
        iterator = iter(iterable)
        while not done_looping:
                try:
                        item = next(iterator)
                except StopIteration:
                        break
                else:
                        action_to_do(item)
```

Python循环结构：扩展知识

- 迭代器

```
nums = [1, 2, 3]
num_iter = iter(nums)
print(next(num_iter))
print(next(num_iter))
print(next(num_iter))
print(next(num_iter))
```

```
1
2
3
StopIteration异常
```

- 可迭代对象的意义 → 可迭代
- 迭代对象器实际上是遍历可迭代对象的代理
- 迭代器没有长度，它们不能被索引
- 可以使用迭代器来做的唯一有用的事情是将其传递给内置的next函数，或者对其进行循环遍历
- 可以使用 list() 函数将迭代器转换为列表

Python循环结构：扩展知识

- 迭代器

```
nums = [1, 2, 3]
num_iter = iter(nums)
print(next(num_iter))
print(list(num_iter))
print(list(num_iter))
```

```
1
[2, 3]
[]
```

- 迭代器是惰性的，只能使用一次，只能循环遍历一次
- 在我们调用 next() 函数之前，它不会做任何事情
- 我们可以创建无限长的迭代器，而创建无限长的列表则不行

Python循环结构：扩展知识

- 迭代器耗尽

```
nums = [1, 2, 3]
squares = [n**2 for n in nums]
print(list(squares))
print(sum(squares))
```

```
[1, 4, 9]
14
```

```
nums = [1, 2, 3]
squares = (n**2 for n in nums)
print(list(squares))
print(sum(squares))
```

```
[1, 4, 9]
0
```

```
nums = [1, 2, 3]
squares = [n**2 for n in nums]
print(sum(squares))
print(tuple(squares))
```

```
14
(1, 4, 9)
```

```
nums = [1, 2, 3]
squares = (n**2 for n in nums)
print(sum(squares))
print(tuple(squares))
```

```
14
()
```

Python循环结构：扩展知识

- 迭代器部分耗尽

```
nums = [1, 2, 3]
squares = (n**2 for n in nums)
print(4 in squares)
print(9 in squares)
```

True
True

```
nums = [1, 2, 3]
squares = (n**2 for n in nums)
print( 9 in squares)
print( 4 in squares)
```

True
False

```
nums = [1, 2, 3]
squares = (n**2 for n in nums)
print( 9 in squares and 4 in squares)
print( 4 in squares)
```

False
False

Python循环结构：扩展知识

- **可迭代**意味着可遍历，通常可以使用for循环遍历
- **序列**是可迭代的，一般从 0 开始索引，索引长度不超过序列长度，可切片
- **可迭代 ≠ 序列**
  - 集合、字典、文件和生成器，一般可统称为“序列结构”

- **迭代器**用于驱动可迭代对象的遍历

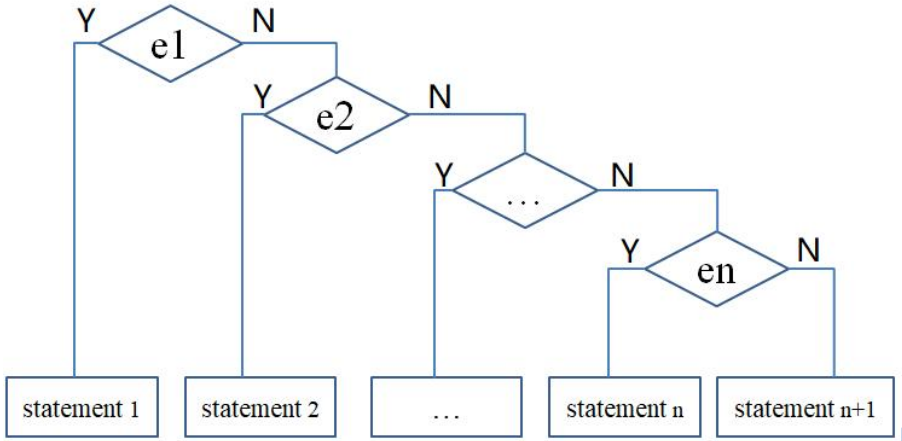
Python循环结构：扩展知识

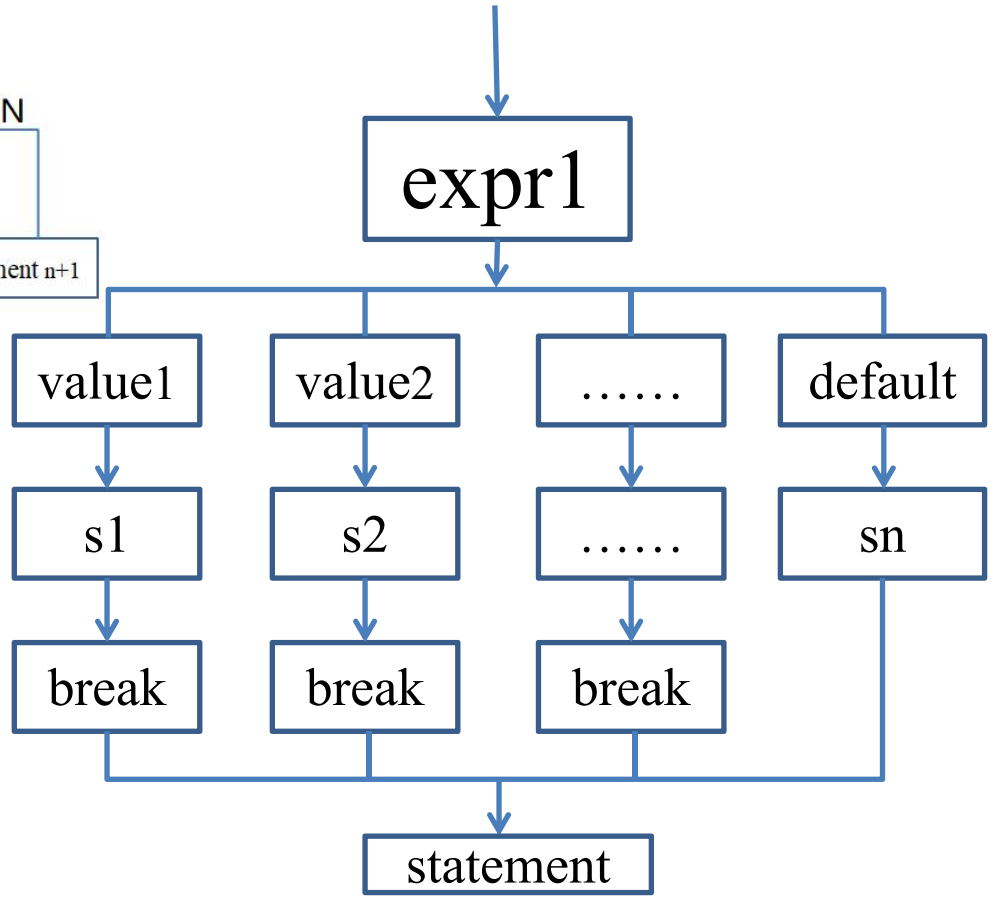
对象	可迭代?	迭代器?
可迭代对象	√	不一定
迭代器	√	√
生成器	√	√
列表	√	×

Python 中有许多迭代器，生成器是迭代器，Python 的许多内置类型也是迭代器。例如，Python 的 enumerate 和 reversed 对象就是迭代器。zip, map 和 filter 也是迭代器；文件对象也是迭代器。

Python选择与循环结构：练习

- C++语言中除了if-else，还有什么选择结构?





Python选择与循环结构：练习

- 利用列表、元组或字典构建跳转表，实现多分支选择功能

```
funcDict = {'1':lambda:print('You input 1'),
            '2':lambda:print('You input 2'),
            '3':lambda:print('You input 3')}
x = input("Input an integer to call a func:")
func = funcDict.get(x, None)
if func:
    func()
else:
    print('Wrong input!')
```

```
Input an integer to call a func:1
You input 1
```

Python选择与循环结构：练习

- 例：面试资格的确认

```
age = 24
subject = '人工智能'
college = '一流大学'

if (age > 25 and subject == '数学') or \
   (college == '一流大学' and subject == '数学') or \
   (age <= 28 and subject =='人工智能'):
    print ("恭喜您获得面试资格！")
else:
    print("抱歉，您未达到面试要求")
```

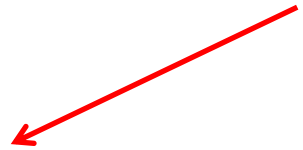
Python选择与循环结构：练习

- 为了优化程序以获得更高的效率和运行速度，在编写循环语句时，应尽量减少循环内部不必要的计算，将与循环变量无关的代码尽可能地提取到循环之外
- 对于使用多重循环嵌套的情况，应尽量减少内层循环中不必要的计算，尽可能地向外提

Python选择与循环结构：练习

```
digits = (1, 2, 3, 4)
for i in range(1000):
    result = []
    for i in digits:
        for j in digits:
            for k in digits:
                result.append(i * 100 + j * 10 + k)
print(result)
```

计算集中在最内层循环



```
[111, 112, 113, 114, 121, 122, 123, 124, 131, 132, 133, 134, 141, 142, 143, 144,
211, 212, 213, 214, 221, 222, 223, 224, 231, 232, 233, 234, 241, 242, 243, 244,
311, 312, 313, 314, 321, 322, 323, 324, 331, 332, 333, 334, 341, 342, 343, 344,
411, 412, 413, 414, 421, 422, 423, 424, 431, 432, 433, 434, 441, 442, 443, 444]
```

Python选择与循环结构：练习

- 将内层的计算向外提

```
digits = (1, 2, 3, 4)
for i in range(1000):
    result = []
    for i in digits:
        for j in digits:
            for k in digits:
                result.append(i * 100 + j * 10 + k)
print(result)
```

运行时间2.41秒

```
digits = (1, 2, 3, 4)
for i in range(100000):
    result = []
    for i in digits:
        i = i * 100
        for j in digits:
            j = j * 10
            for k in digits:
                result.append(i + j + k)
print(result)
```

运行时间2.03秒

Python选择与循环结构：练习

- 循环中尽量引用局部变量，因为局部变量的查询和访问速度比全局变量略快
- 使用模块中的方式时，可以将其直接导入来减少查询次数，提高运行速度

```
import math
for i in range(100000):
    math.sin(i)
```

运行时间0.026秒

```
import math
local_sin = math.sin
for i in range(100000):
    local_sin(i)
```

运行时间0.022秒

Python选择与循环结构：练习

- 警惕continue可能带来的问题：

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

输出1 3 5 7 9

```
for i in range(10):
    if i % 2 == 0:
        i = i + 1    #没用!
        continue
    print(i)
```

输出1 3 5 7 9

- 每次进入循环时的 i 已经不是上一次的 i 了

Python选择与循环结构：练习

- 输出序列中的元素

```
a_list = ['a', 'b', 'seu', 'jiao8', '201']
for i, v in enumerate(a_list):
    print('列表的第', i + 1, '个元素是: ', v)
```

```
列表的第 1 个元素是:  a
列表的第 2 个元素是:  b
列表的第 3 个元素是:  seu
列表的第 4 个元素是:  jiao8
列表的第 5 个元素是:  201
```

Python选择与循环结构：练习

- 打印九九乘法表

```
for i in range(1,10):
    for j in range(1,i + 1):
        print('{0}*{1}={2}'.format(i, j, i * j).ljust(6), \
              end = ' ')
    print()
```

```
1*1=1
2*1=2  2*2=4
3*1=3  3*2=6  3*3=9
4*1=4  4*2=8  4*3=12 4*4=16
5*1=5  5*2=10 5*3=15 5*4=20 5*5=25
6*1=6  6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
7*1=7  7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
8*1=8  8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
9*1=9  9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

Python选择与循环结构：练习

- 鸡兔同笼问题：假设共有鸡、兔30只，脚90只，问鸡和兔各有多少只?

```
for tutu in range(31):
    if tutu*4 + (30-tutu)*2 == 90:
        print('鸡:',30-tutu,'兔:',tutu)
```

Python选择与循环结构：练习

- 使用列表推导式实现嵌套列表的平铺

```
vec = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
a_List = [num for elem in vec \
          for num in elem]
```

输出a_List=[1, 2, 3, 4, 5, 6, 7, 8, 9]

等价于：

```
vec = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
a_List = []
for elem in vec:
    for num in elem:
        a_List.append(num)
```

Python选择与循环结构：练习

- 多变量赋值

```
>>> a, b = 0, 1
>>> a
0
>>> b
1
```

```
>>> a, b = 0, 1
>>> a, b = b, a
>>> a
1
>>> b
0
```

```
>>> a = 2
>>> a, b = 1, a
>>> a
1
>>> b
2
```

存的是指向2的引用

```
>>> (r, g, b) = ["Red","Green","Blue"]
>>> r
'Red'
>>> g
'Green'
>>> b
'Blue'
```

Python选择与循环结构：练习

- 多变量赋值又称“序列解包”

```
>>> x, y, z = 1, 2, 3
>>> a_tuple = (1, 2, 3)
>>> (x, y, z) = a_tuple
>>> x, y, z = a_tuple
>>> x, y, z = range(3)   #对range对象进行序列解包

>>> x, y, z = iter([1,2,3])      #对迭代器对象进行序列解包
>>> x, y, z = map(int, range(3))#对map对象进行序列解包
>>> x, y, z = sorted([1, 3, 2]) #sorted()方法返回的列表进行解包
>>> x, y, z = '123'
>>> x = [1, 2, 3, 4, 5]
>>> x[:3] = map(str, range(3))   #切片也支持序列解包
>>> x
['0', '1', '2', 4, 5]
```

Python选择与循环结构：练习

- 序列解包是迭代，当直接在字典上迭代时，得到的是键

```
counts = {1:'a', 2:'b'}
x, y = counts
print(x, y)
```

1 2

作业题

- 整数反转
  - 给出一个有符号整数，将每位上的数字进行反转。例如，输入: 123，输出: 321；输入: -123，输出: -321
  - 实现时不能用列表、字符串等序列结构
- 无重复字符的最长字串
  - 给定一个字符串，找出其中不含有重复字符的最长子串的长度。例如，输入：“pwwkew”，输出：3
- 集合的笛卡尔乘积
  - 给定两个集合X和Y，假设X={a,b}, Y={0,1,2}, 它们的笛卡尔积为{(a,0), (a,1), (a,2), (b,0), (b,1), (b, 2)} （用推导式实现）

作业题

- 模拟投骰子
  - 投100次骰子，统计1~6出现的次数
  - 投1000次骰子，统计连续两个6出现的次数（如第一次为6，第二次也为6，则出现1次，第三次还为6，则出现2次，以此类推），估算连续两个6出现的概率
  - 投1000次骰子，统计某个数比如6连续出现的最大次数，并估算这个数的概率分布

  （前2题不用for循环，第2题可用列表解析/列表推导式实现）