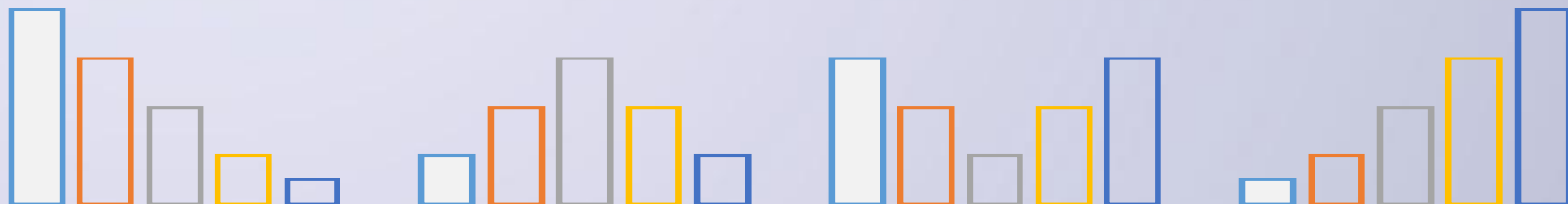


# Python语言程序设计

成都信息工程大学区块链产业学院

刘硕

# 第4章 程序的控制结构及 选择与循环



# 前情回顾

- 温度转换代码
- Turtle小海龟代码
- 数据类型和math库
- 文本进度条和time库

# 目录

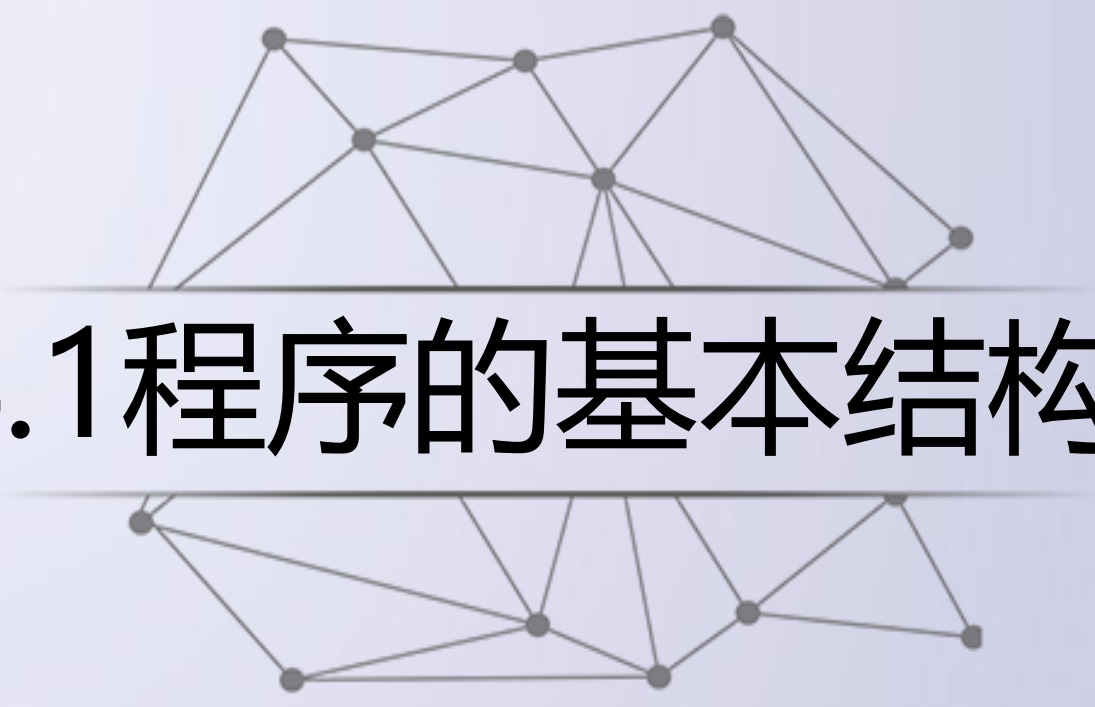
**4.1 程序的基本结构**

**4.2 程序的分支结构**

**4.3 程序的循环结构**

**4.4 random库的使用**

**4.5  $\pi$ 的计算**

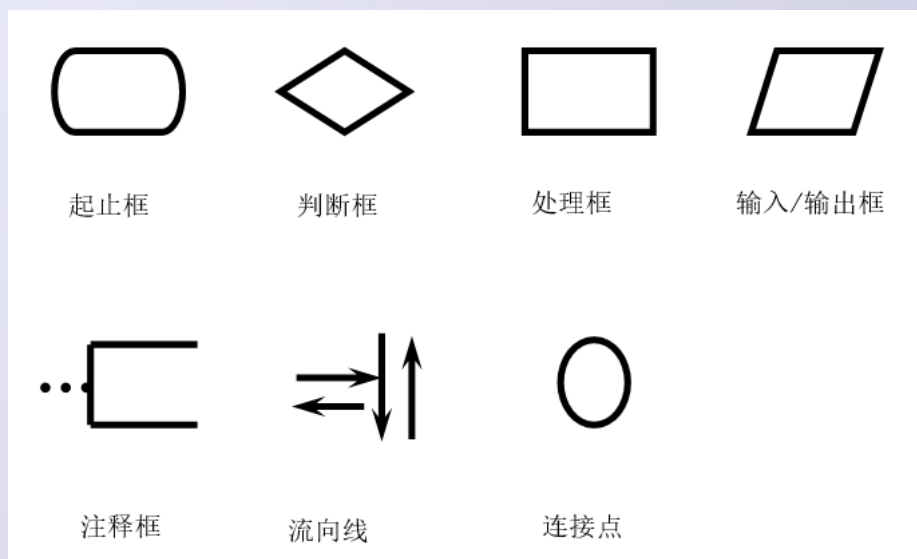


## 4.1 程序的基本结构

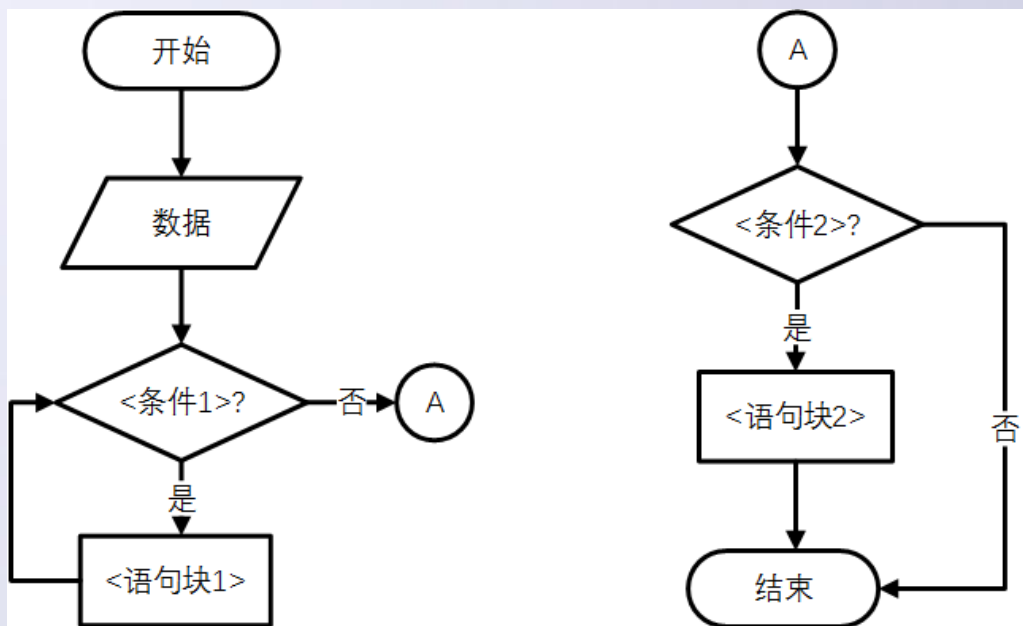
# 程序的流程图

程序流程图用一系列图形、流程线和文字说明描述程序的基本操作和控制流程，它是程序分析和过程描述的最基本方式。

• 流程图的基本元素包括7种



# 程序的流程图



程序流程图示例：由连接点A连接的一个程序



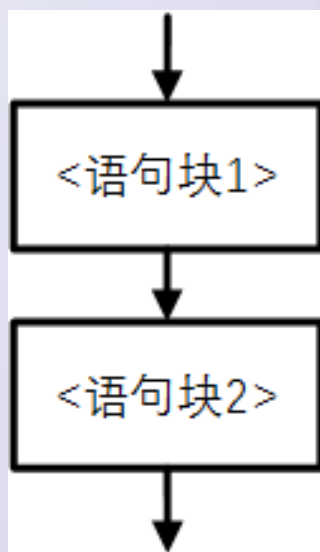
# 程序的基本结构

- 顺序结构是程序的基础，但单一的顺序结构不可能解决所有问题。
- 程序由三种基本结构组成：
  - 顺序结构
  - 分支结构
  - 循环结构
- 这些基本结构都有一个入口和一个出口。任何程序都由这三种基本结构组合而成



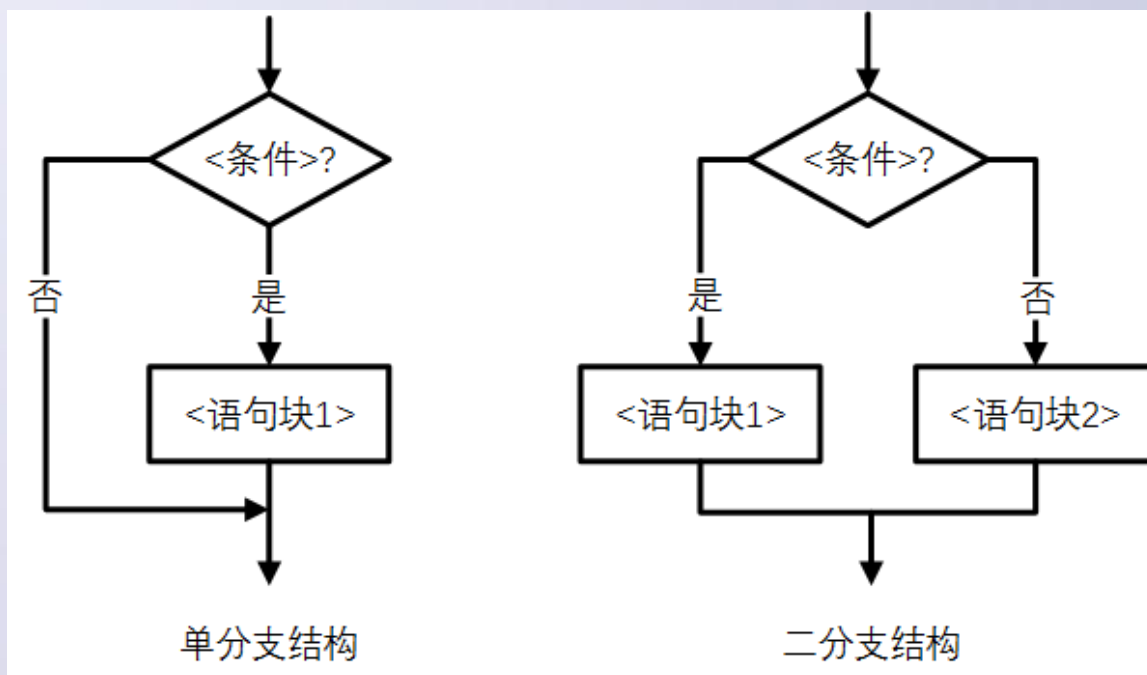
# 程序的基本结构

- 顺序结构是程序按照线性顺序依次执行的一种运行方式，其中语句块1和语句块2表示一个或一组顺序执行的语句



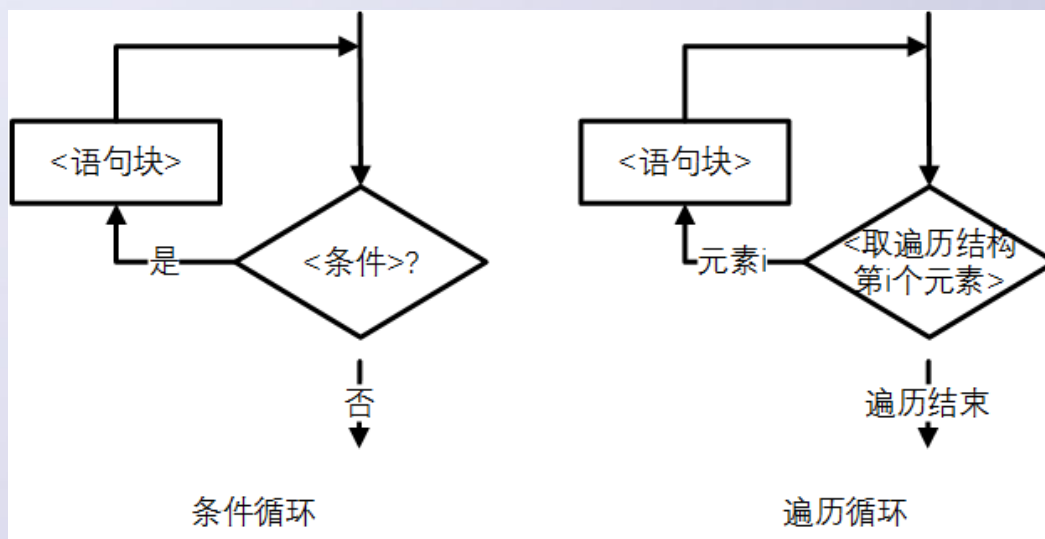
# 程序的基本结构

分支结构是程序根据条件判断结果而选择不同向前执行路径的一种运行方式，包括单分支结构和二分支结构。由二分支结构会组合形成多分支结构



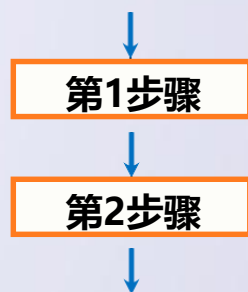
# 程序的基本结构

循环结构是程序根据条件判断结果向后反复执行的一种运行方式，根据循环体触发条件不同，包括条件循环和遍历循环结构

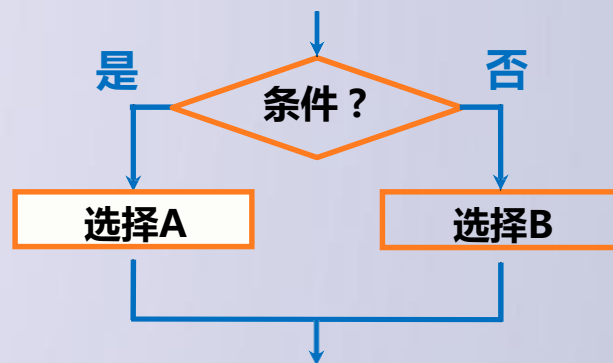


## “程序的控制结构”

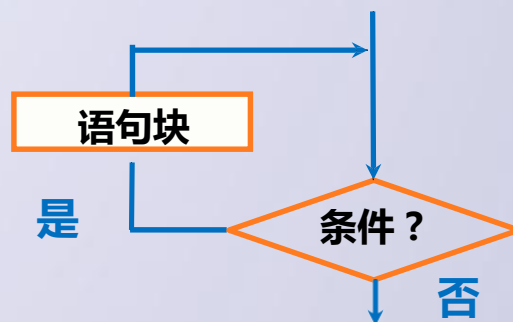
- 顺序结构



- 分支结构



- 循环结构



# 程序的基本结构实例

对于一个计算问题，可以用IPO描述、流程图描述或者直接以Python代码方式描述

微实例：圆面积和周长的计算。

输入：圆半径R

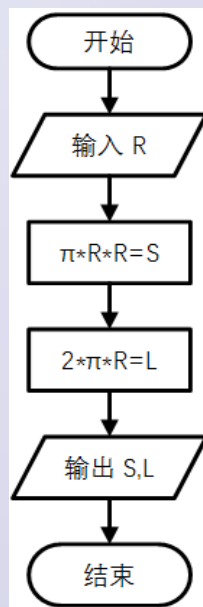
处理：

圆面积： $S = \pi * R * R$

圆周长： $L = 2 * \pi * R$

输出：圆面积S、周长L

问题IPO描述



```
1  R = eval(input("请输入圆半径:"))
2  S = 3.1415*R*R
3  L = 2*3.1415*R
4  print("面积和周长:",S,L)
```

Python代码描述

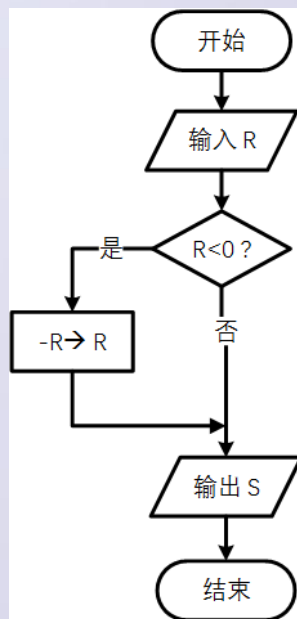
# 程序的基本结构实例

微实例2：实数绝对值的计算。

输入：实数R

$$\text{处理: } |R| = \begin{cases} R & R \geq 0 \\ -R & R < 0 \end{cases}$$

输出：输出|R|



```
1  R = eval(input("输入实数:"))
2  if (R < 0):
3      R = -R
4  print("绝对值",R)
```

(a) 问题IPO描述

(b) 流程图描述

(c) Python代码描述

# 程序的基本结构实例

## 微实例3：整数累加。

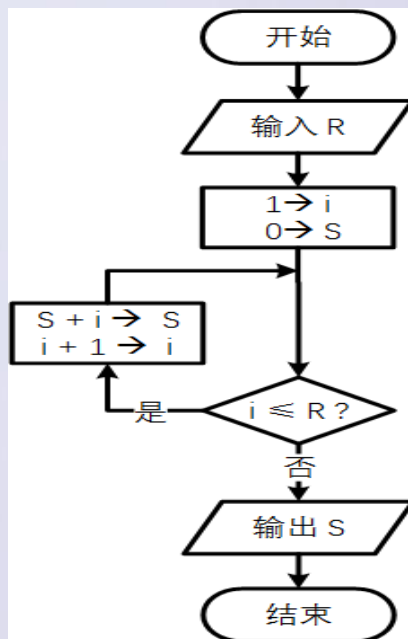
输入：正整数R

处理：

$$S=1+2+3+\cdots+R$$

输出：输出S

(a) 问题IPO描述



(b) 流程图描述

```
1  R = eval(input("请输入正
2  整数:"))
3
4  i, S = 0, 0
5  while (i<=R):
6      S = S + i
7      i = i + 1
8  print("累加求和",S)
```

(c) Python代码描述



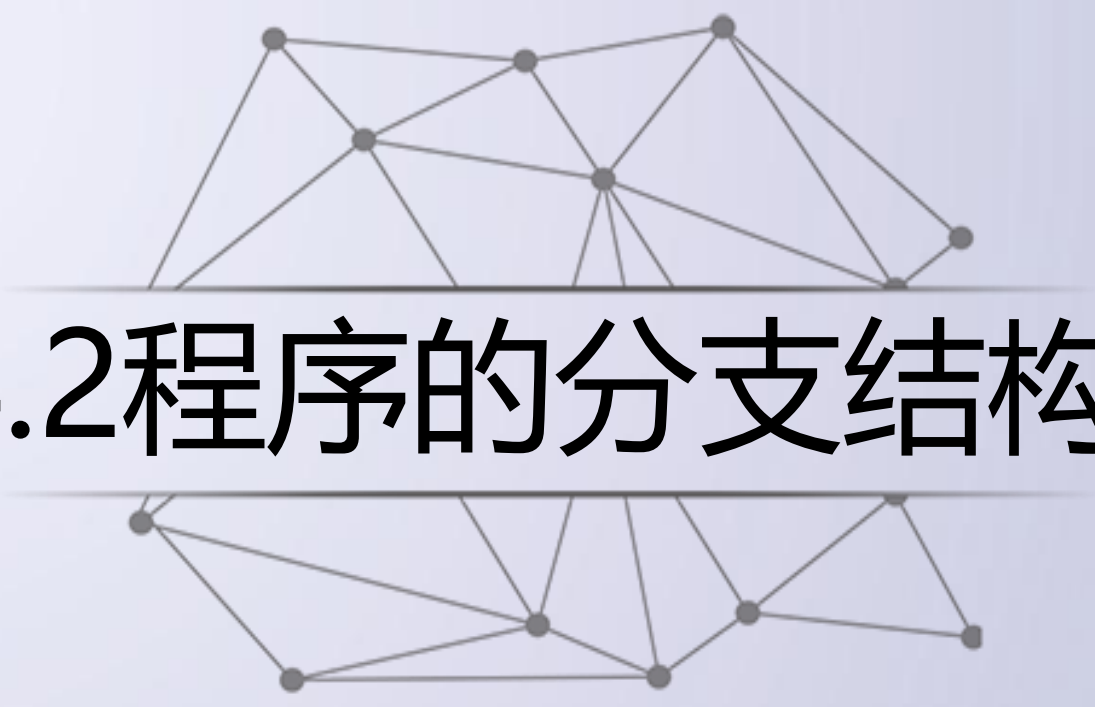
# 程序的基本结构实例

IPO描述主要用于区分程序的输入输出关系，重点在于结构划分，算法主要采用自然语言描述


流程图描述侧重于描述算法的具体流程关系，流程图的结构化关系相比自然语言描述更进一步，有助于阐述算法的具体操作过程

Python代码描述是最终的程序产出，最为细致。





## 4.2程序的分支结构



# 单分支结构: if语句

Python中if语句的语法格式如下:

if <条件>:

语句块

- 语句块是if条件满足后执行的一个或多个语句序列
- 语句块中语句通过与if所在行形成缩进表达包含关系
- if语句首先评估<条件>的结果值, 如果结果为True, 则执行语句块里的语句序列, 然后控制转向程序的下一条语句。如果结果为False, 语句块里的语句会被跳过。

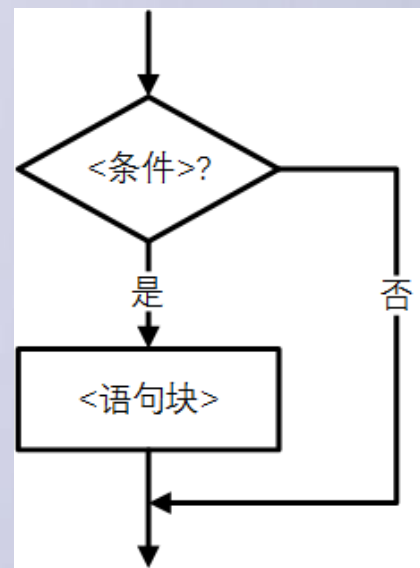
# 单分支结构: if语句

if语句中语句块执行与否依赖于条件判断。但无论什么情况，控制都会转到if语句后与该语句同级别的下一条语句


```
x = input('Input two number:')
a, b = map(int, x.split())
if a > b:
    a, b = b, a      #序列解包，交换两个变量的值
    print(a, b)
```

```
===== while True:
Input two numbers:3 4      x = input('Input two numbers:')
Input two numbers:4 3      a, b = map(int, x.split())
3 4                         if a > b:
Input two numbers:         a, b = b, a
                           print(a, b)
```

```
===== while True:
Input one numbers:3      a = eval(input('Input one numbers:'))
Input one numbers:4      b = eval(input('Input one numbers:'))
Input one numbers:5      if a > b:
Input one numbers:2      a, b = b, a
2 5                       print(a, b)
Input one numbers:
```



if语句的控制流程图



# 单分支结构: if语句

- if语中<条件>部分可以使用任何能够产生True或False的语句
- 形成判断条件最常见的方式是采用关系操作符
- Python语言共有6个关系操作符

| 操作符 | 数学符号 | 操作符含义 |
|-----|------|-------|
| <   | <    | 小于    |
| <=  | ≤    | 小于等于  |
| >=  | ≥    | 大于等于  |
| >   | >    | 大于    |
| ==  | =    | 等于    |
| !=  | ≠    | 不等于   |

# 单分支结构

## 单分支示例

```
guess = eval(input())
```

```
if guess == 99:
```

```
    print("猜对了")
```

```
if True:
```

```
    print("条件正确")
```



# 二分支结构: if-else语句

Python中if-else语句用来形成二分支结构，语法格式如下：

```
if <条件>:  
    <语句块1>  
else:  
    <语句块2>
```

```
while True:  
    x = input('Input two numbers:')  
    a, b = map(int, x.split())  
    if a>b:  
        a, b=b, a  
    else:  
        a, b=a-3, b+3  
    print(a, b)
```

```
=====  
Input two numbers:23 34  
20 37  
Input two numbers:34 23  
23 34  
Input two numbers:
```

- <语句块1>是在if条件满足后执行的一个或多个语句序列
- <语句块2>是if条件不满足后执行的语句序列
- 二分支语句用于区分<条件>的两种可能True或者False，分别形成执行路径

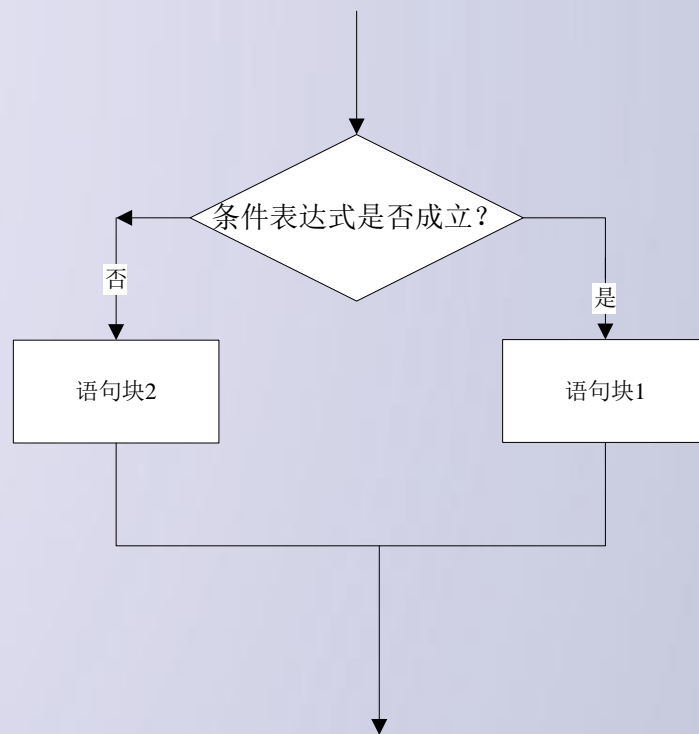
# 二分支结构: if-else语句

```
if 表达式:  
    语句块1  
else:  
    语句块2
```

```
>>> chTest = ['1', '2', '3', '4', '5']
```

```
>>> if chTest:  
    print(chTest)  
else:  
    print('Empty')
```

```
['1', '2', '3', '4', '5']
```



# 二分支结构

## 二分支示例

```
guess = eval(input())
```

```
if guess == 99:
```

```
    print("猜对了")
```

```
else :
```

```
    print("猜错了")
```

```
if True:
```

```
    print("语句块1")
```

```
else :
```

```
    print("语句块2")
```



# 二分支结构

紧凑形式：适用于简单表达式的二分支结构

〈表达式1〉 *if* 〈条件〉 *else* 〈表达式2〉

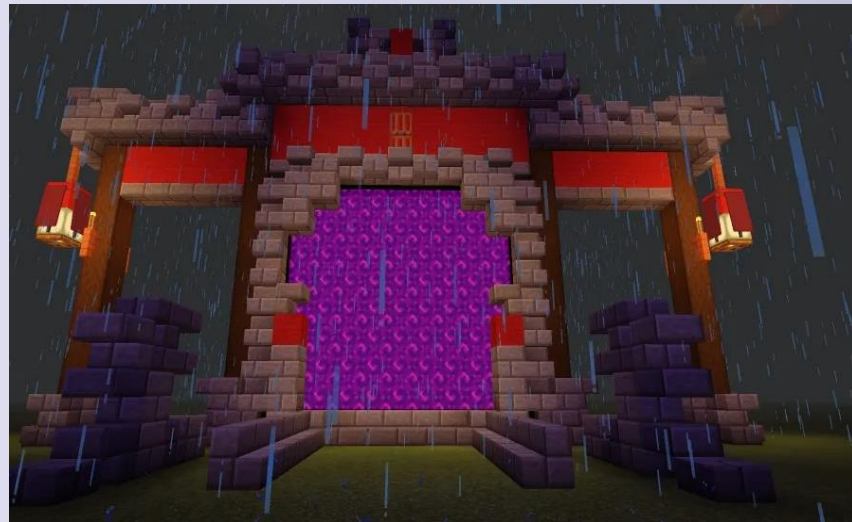
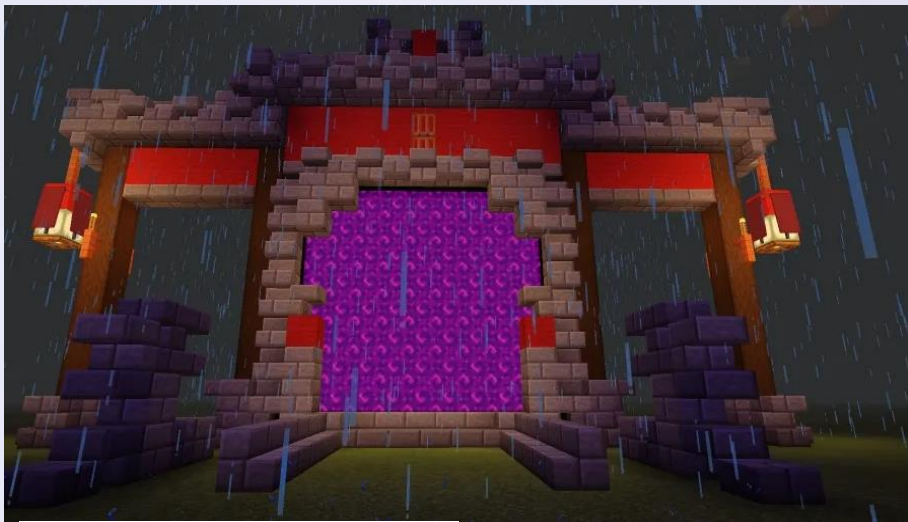
```
guess = eval(input())
```

```
print("猜{}了".format("对" if guess==99 else "错"))
```

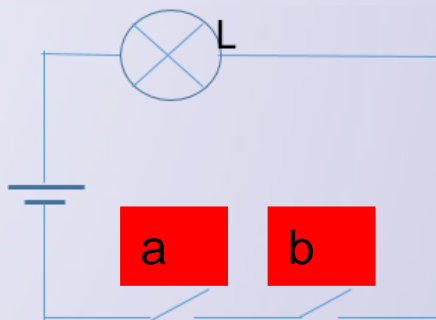
# 二分支结构: if-else语句

if...else的紧凑结构非常适合对特殊值处理的情况，如下：

```
>>>count = 2
>>>count if count!=0 else "不存在"
2
>>>count = 0
>>>count if count!=0 else "不存在"
"不存在"
```



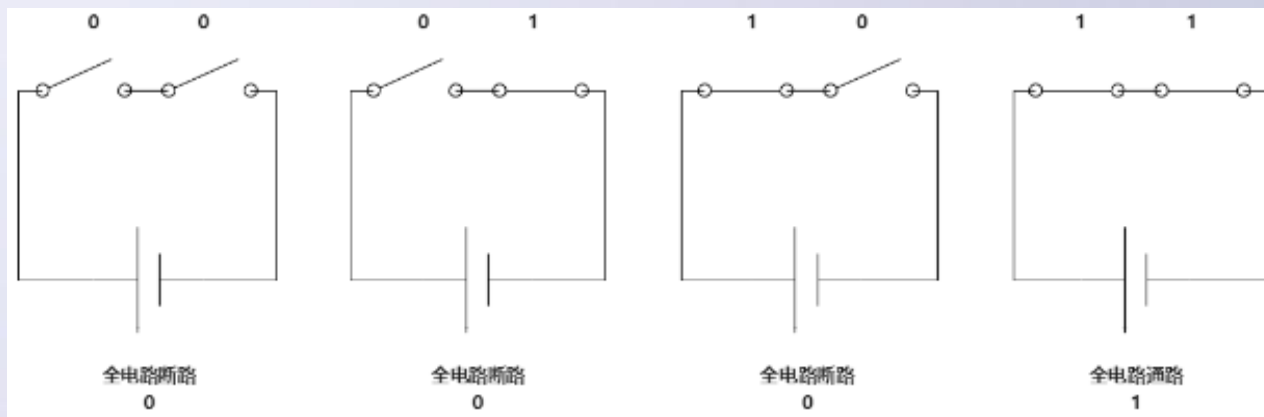
问a 假如问b哪个是天堂的话，他会如何回答？



|           | Honest-b     | Cheater-b    | Cheater-b    |
|-----------|--------------|--------------|--------------|
| Honest-a  |              | b a a-heaven | a b b-heaven |
| Cheater-a | a b b-heaven |              |              |
| Cheater-a | b a a-heaven |              |              |
|           |              |              |              |

## 与门

| a | b | a and b |
|---|---|---------|
| 0 | 0 | 0       |
| 0 | 1 | 0       |
| 1 | 0 | 0       |
| 1 | 1 | 1       |

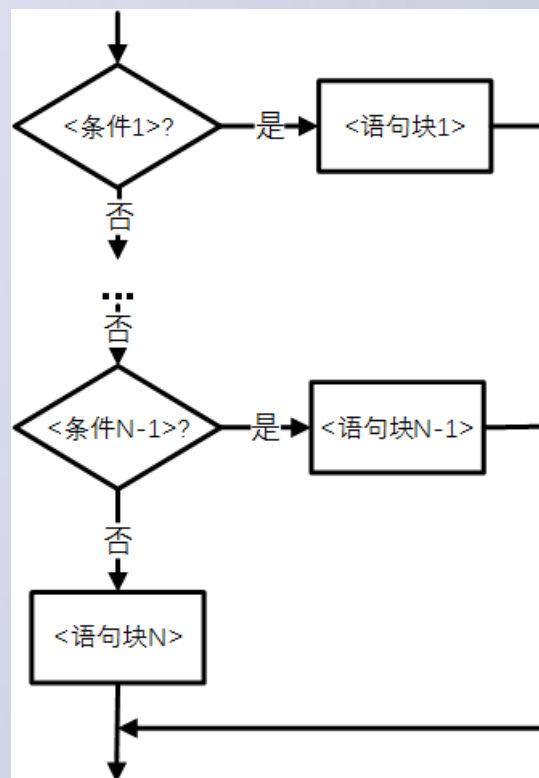


不论问a，还是问b，得到的都是谎言，

# 多分支结构: if-elif-else语句

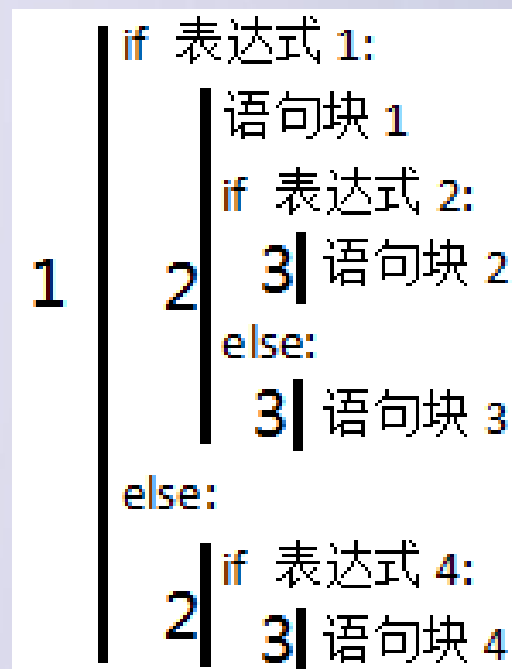
Python的if-elif-else描述多分支结构，语句格式如下：

```
if <条件1>:  
    <语句块1>  
elif <条件2>:  
    <语句块2>  
...  
else:  
    <语句块N>
```



# 多分支结构的选择嵌套

```
if 表达式1:  
    语句块1  
    if 表达式2:  
        语句块2  
    else:  
        语句块3  
else:  
    if 表达式4:  
        语句块4
```



注意：缩进必须要正确并且一致。

# 选择结构的嵌套

- 使用嵌套的选择结构实现百分制成绩到等级制的转换

```
>>> def func(score):  
    degree = 'DCBAE'  
    if score > 100 or score < 0:  
        return 'wrong score.must between 0 and 100.'  
    else:  
        index = (score - 60)//10  
        if index >= 0:  
            return degree[index]  
        else:  
            return degree[-1]
```



# 多分支结构: if-elif-else语句

- 多分支结构是二分支结构的扩展，这种形式通常用于设置同一个判断条件的多条执行路径。
- Python依次评估寻找第一个结果为True的条件，执行该条件下的语句块，同时结束后跳过整个if-elif-else结构，执行后面的语句。如果没有任何条件成立，else下面的语句块被执行。else子句是可选的



# 小练习

关于多分支结构，那个选项的描述是正确的？

**A**多分支结构是使用最广泛的程序控制结构，可替代任何分支结构

**B**多分支结构是二分支结构的扩展

**C**多分支结构仅用来根据完全不相关的多种判断条件设置多条执行路径

**D**多分支结构采用if-elif-else描述，其中elif和else都是可选的

**B**

---

# 多分支结构

## 对不同分数分级的问题

```
score = eval(input())
```

```
if score >= 60:
```

```
    grade = "D"
```

```
elif score >= 70:
```

```
    grade = "C"
```

```
elif score >= 80:
```

```
    grade = "B"
```

```
elif score >= 90:
```

```
    grade = "A "
```

```
print("输入成绩属于级别{}".format(grade))
```

— 注意多条件之间的包含关系

— 注意变量取值范围的覆盖

# 多分支结构

- 利用多分支选择结构将成绩从百分制变换到等级制

```
def func(score):
```

```
    if score > 100:
```

```
        return 'wrong score.must <= 100.'
```

```
    elif score >= 90:
```

```
        return 'A'
```

```
    elif score >= 80:
```

```
        return 'B'
```

```
    elif score >= 70:
```

```
        return 'C'
```

```
    elif score >= 60:
```

```
        return 'D'
```

```
    elif score >= 0:
```

```
        return 'E'
```

```
    else:
```

```
        return 'wrong score.must >0'
```

**小练习-P107 4.10**



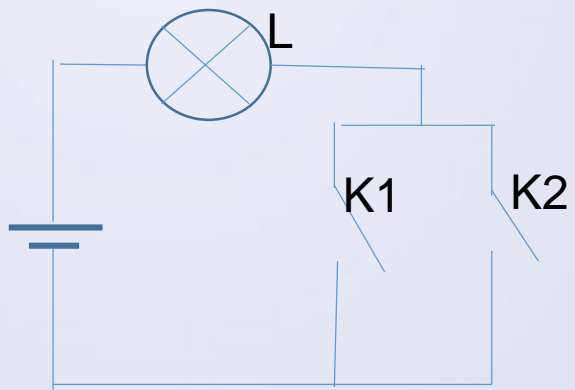
# 条件判断及组合

# 条件判断

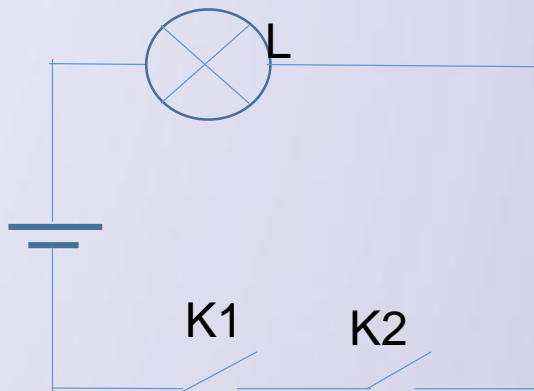
操作符

| 操作符 | 数学符号 | 描述   |
|-----|------|------|
| <   | <    | 小于   |
| <=  | ≤    | 小于等于 |
| >=  | ≥    | 大于等于 |
| >   | >    | 大于   |
| ==  | =    | 等于   |
| !=  | ≠    | 不等于  |

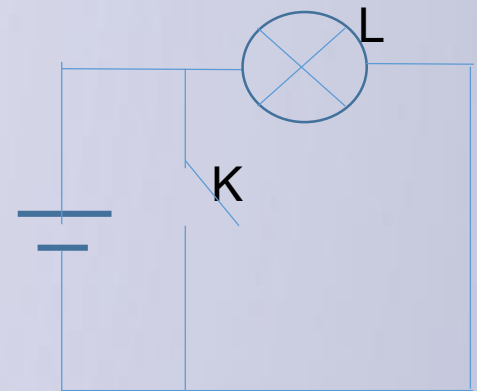
- 逻辑运算符and和or以及关系运算符具有惰性求值特点



(1) or, 并联电路



(2) and, 串联电路



(3) not, 短路

# 惰性求值的几点说明

- 比较特殊的运算符还有逻辑运算符“and”和“or”以及关系运算符，具有短路求值或惰性求值的特点，只计算必须计算的表达式的值。
- 以“and”为例，对于表达式“表达式1 and 表达式2”而言，如果“表达式1”的值为“False”或其他等价值时，不论“表达式2”的值是什么，整个表达式的值都是“False”，此时“表达式2”的值无论是什么都不影响整个表达式的值，因此将不会被计算，从而减少不必要的计算和判断。

# 效率就是第一

- 在设计条件表达式时，如果能够大概预测不同条件失败的概率，并将多个条件根据“and”和“or”运算的短路求值特性进行组织，可以大幅度提高程序运行效率。下面的函数用来使用用户指定的分隔符将多个字符串连接成一个字符串，如果用户没有指定分隔符则使用逗号。

```
>>> def Join(chList, sep=None):  
    return (sep or ',').join(chList)
```

```
>>> chTest = ['1', '2', '3', '4', '5']
```

```
>>> Join(chTest)
```

```
'1,2,3,4,5'
```

```
>>> Join(chTest, ':')
```

```
'1:2:3:4:5'
```

```
>>> Join(chTest, ' ')
```

```
'1 2 3 4 5'
```

默认为none  
，下一个条  
件应该为  
True



# math random: 谁会是多余的?

```
>>> x = math.sqrt(9) if 5>3 else random.randint(1, 100)
```

```
>>> x = math.sqrt(9) if 5>3 else random.randint(1, 100)
...
Traceback (most recent call last):
  File "<pyshell#92>", line 1, in <module>
    x = math.sqrt(9) if 5>3 else random.randint(1, 100)
NameError: name 'math' is not defined
```

#此时还没有导入math模块

```
>>> import math
```

#此时还没有导入random模块, 但由于条件表达式 $5>3$ 的值为True, 所以可以正常运行



```
>>> x = math.sqrt(9) if 2>3 else random.randint(1, 100)
```

```
NameError: name 'random' is not defined
```

#此时还没有导入random模块, 由于条件表达式 $2>3$ 的值为False, 需要计算第二个表达式的值, 因此出错

```
>>> import random
```

```
>>> x = math.sqrt(9) if 2>3 else random.randint(1, 100)
```

# 条件判断及组合

## 示例


```
guess = eval(input())  
if guess > 99 or guess < 99:  
    print("猜错了")  
else :  
    print("猜对了")
```

# 条件判断及组合

- 另外，在Python中，条件表达式中不允许使用赋值运算符“=”，避免了其他语言中误将关系运算符“==”写作赋值运算符“=”带来的麻烦，例如下面的代码，在条件表达式中使用赋值运算符“=”将抛出异常，提示语法错误。

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

```
>>> if (a=3) and (b=4):  
SyntaxError: invalid syntax
```



# 程序的异常处理

# 异常处理

```
num = eval(input("请输入一个整数: "))  
print(num**2)
```

**当用户没有输入整数时，会产生异常，怎么处理？**

# 异常处理

## 异常发生的代码行数

Traceback (most recent call last):

File "t.py", line 1, in <module>

num = eval(input("请输入一个整数: "))

File "<string>", line 1 in

<module>

NameError: name 'abc' is not defined

异常类型

异常内容提示

# 异常处理

## 异常处理的基本使用

*try* :

＜语句块1＞

*except* :

＜语句块2＞

*try* :

＜语句块1＞

*except* ＜异常类型＞ :

＜语句块2＞

# 异常处理

## 示例1

*try :*

```
num = eval(input("请输入一个整数: "))
```

```
print(num**2)
```

*except :*

```
print("输入不是整数")
```



# 异常处理

## 示例2

*try :*

```
num = eval(input("请输入一个整数: "))  
print(num**2)
```

*except* NameError:

标注异常类型后，仅响应此类异常

```
print("输入不是整数")
```

异常类型名字等同于变量名

# 异常处理

## 异常处理的高级使用

*try* :

＜语句块1＞

*except* :

＜语句块2＞


*else* :

＜语句块3＞

*finally* :

＜语句块4＞

- *else* 对应语句块3在不发生异常时执行
- *finally* 对应语句块4一定执行



# 异常处理: try-except语句

观察下面这段小程序:

```
1 num = eval(input("请输入一个整数: "))
2 print(num**2)
```

当用户输入的不是数字呢?

```
>>>
请输入一个整数: 100
10000
>>>
请输入一个整数: NO
Traceback (most recent call last):
  File "D:/PythonPL/echoInt.py", line 1, in <module>
    num = eval(input("请输入一个整数: "))
  File "<string>", line 1, in <module>
NameError: name 'No' is not defined
```

# 异常处理: try-except语句

Python解释器返回了异常信息，同时程序退出

```
Traceback (most recent call last):  
File "D:/PythonPL/echoInt.py", line 1, in <module>  
    num = eval(input("请输入一个整数: "))  
File "<string>", line 1, in <module>  
NameError: name 'No' is not defined
```

异常回溯标记

异常文件路径

异常发生的代码行数

异常类型

异常内容提示



# 异常处理: **try-except**语句


- Python异常信息中最重要的部分是异常类型，它表明了发生异常的原因，也是程序处理异常的依据。
- Python使用try-except语句实现异常处理，基本的语法格式如下：

```
try:
```

```
    <语句块1>
```

```
except <异常类型>:
```

```
    <语句块2>
```



# 异常处理: try-except语句

```
1 try:
2     num = eval(input("请输入一个整数: "))
    print(num**2)
except NameError:
    print("输入错误, 请输入一个整数!")
```

该程序执行效果如下:

```
>>>
请输入一个整数: NO
输入错误, 请输入一个整数!
```



# 异常的高级用法

- try-except语句可以支持多个except语句，语法格式如下：

try:

    <语句块1>

except <异常类型1>:

    <语句块2>

....

except <异常类型N>:

    <语句块N+1>

except:

    <语句块N+2>



# 异常的高级用法

- 最后一个except语句没有指定任何类型，表示它对应的语句块可以处理所有其他异常。这个过程与if-elif-else语句类似，是分支结构的一种表达方式，一段代码如下

```
1  try:
2      alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
3      idx = eval(input("请输入一个整数: "))
4      print(alp[idx])
5  except NameError:
6      print("输入错误, 请输入一个整数!")
7  except:
8      print("其他错误")
```





# 异常的高级用法

该程序将用户输入的数字作为索引从字符串alp中返回一个字符，当用户输入非整数字符时，`except NameError`异常被捕获到，提示升用户输入类型错误，当用户输入数字不在0到25之间时，异常被`except`捕获，程序打印其他错误信息，执行过程和结果如下：

```
>>>
```

```
请输入一个整数： NO  
输入错误，请输入一个整数！
```

```
>>>
```

```
请输入一个整数： 100  
其他错误
```



# 异常的高级用法

除了try和except保留字外，异常语句还可以与else和finally保留字配合使用，语法格式如下：

try:

    <语句块1>

except <异常类型1>:

    <语句块2>

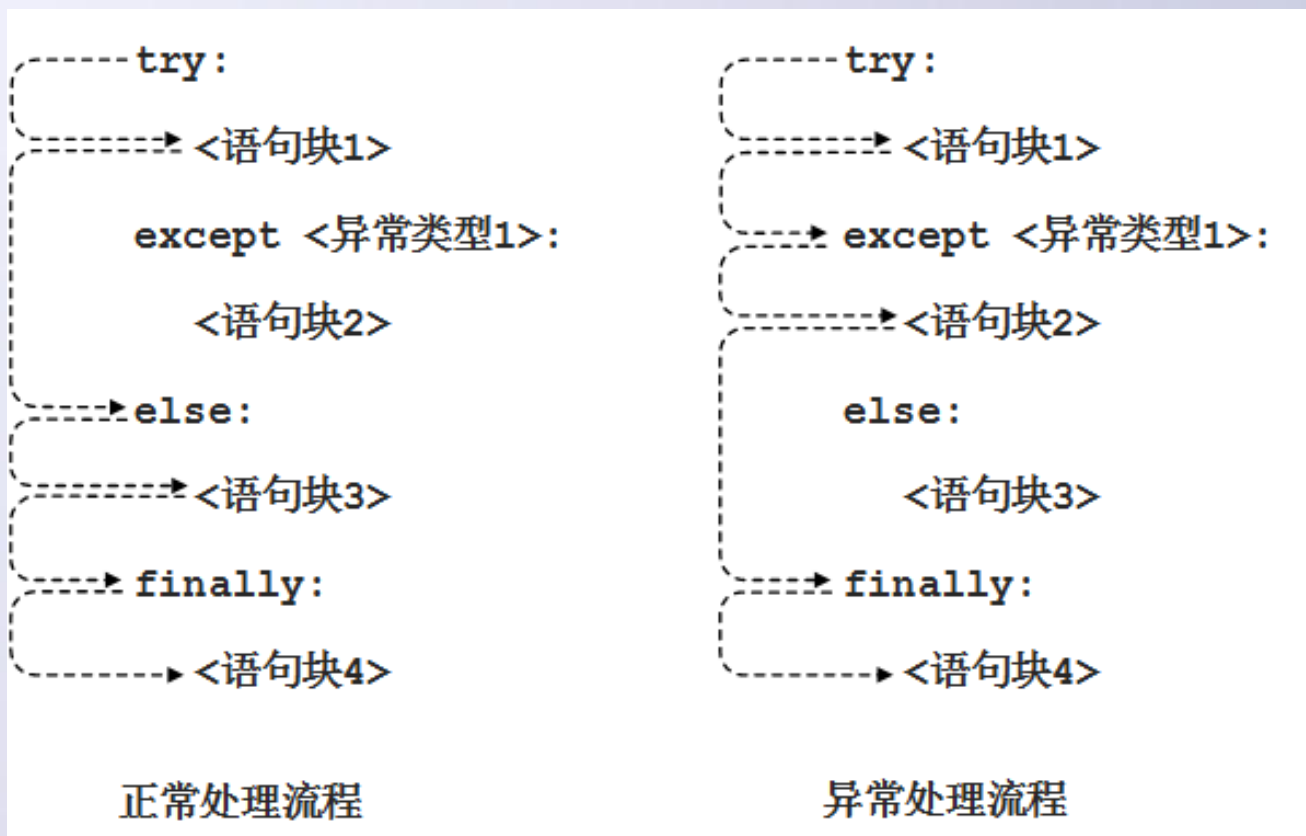
else:

    <语句块3>

finally:

    <语句块4>

# 异常的高级用法





# 异常的高级用法

采用else和finally修改代码如下：

```
1  try:
2      alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
3      idx = eval(input("请输入一个整数: "))
4      print(alp[idx])
5  except NameError:
6      print("输入错误, 请输入一个整数!")
7  else:
8      print("没有发生异常")
9  finally:
10     print("程序执行完毕, 不知道是否发生了异常")
```

执行过程和结果如下：

```
>>>
```

请输入一个整数： 5

```
F
```

没有发生异常


程序执行完毕，不知道是否发生了异常

```
>>>
```

请输入一个整数： NO

输入错误，请输入一个整数！

程序执行完毕，不知道是否发生了异常



# 单元小结

# 程序的分支结构

- 单分支 *if* 二分支 *if-else* 及紧凑形式
- 多分支 *if-elif-else* 及条件之间关系
- *not and or*      *> >= == <= < !=*
- 异常处理      *try-except-else-finally*



# 身体质量指数BMI

BMI的定义如下:  $BMI = \text{体重 (kg)} \div \text{身高}^2 \text{ (m}^2\text{)}$

例如NBA勇士队的当家球星库里身高1.91米,  
体重90.7公斤,  
他是否健康呢?

凯文杜兰特身高2.11米,  
体重106.6公斤,  
是否健康?



# 身体质量指数BMI

编写一个根据体重和身高计算BMI值的程序，并同时输出国际和国内的BMI指标建议值

| 分类 | 国际BMI值 (kg/m <sup>2</sup> ) | 国内BMI值 (kg/m <sup>2</sup> ) |
|----|-----------------------------|-----------------------------|
| 偏瘦 | < 18.5                      | < 18.5                      |
| 正常 | 18.5 ~ 25                   | 18.5 ~ 24                   |
| 偏胖 | 25 ~ 30                     | 24 ~ 28                     |
| 肥胖 | >= 30                       | >= 28                       |

库里的BMI = 体重 (kg) ÷ 身高<sup>2</sup> (m<sup>2</sup>) = 24.86





# 身体质量指数

## BMI

### 问题需求

- 输入：给定体重和身高值
- 输出：BMI指标分类信息(国际和国内)



# "身体质量指数BMI"实例讲解

# 身体质量指标BMI

## 思路方法

—难点在于同时输出国际和国内对应的分类

—思路1：分别计算并给出国际和国内BMI分类

—思路2：混合计算并给出国际和国内BMI分类

# 身体质量指标BMI

```
#CalBMIV1.py
```

```
height, weight = eval(input("请输入库里的身高(米)和体重(公斤)[逗号隔开]:"))
```

```
bmi = weight / pow(height, 2)
```

```
print("BMI 数值为: {:.2f}".format(bmi))
```

```
who = ""
```

```
if bmi < 18.5:
```

```
    who = "偏瘦"
```

```
elif 18.5 <= bmi < 25:
```

```
    who = "正常"
```

```
elif 25 <= bmi < 30:
```

```
    who = "偏胖"
```

```
else:
```

```
    who = "肥胖"
```

```
print("BMI 指标为:国际'{0}'".format(who))
```

| 分类 | 国际BMI值    | 国内BMI值    |
|----|-----------|-----------|
| 偏瘦 | <18.5     | <18.5     |
| 正常 | 18.5 ~ 25 | 18.5 ~ 24 |
| 偏胖 | 25 ~ 30   | 24 ~ 28   |
| 肥胖 | ≥30       | ≥28       |

# 身体质量指标

## BMI

```
#CalBMIV2.py
```

```
height, weight = eval(input("请输入库里的身高(米)和体重(公斤)[逗号隔开]: "))
```

```
bmi = weight / pow(height, 2)
```

```
print("BMI 数值为: {:.2f}".format(bmi))  nat = ""
```

```
if bmi < 18.5:
```

```
    nat = "偏瘦"
```

```
elif 18.5 <= bmi < 24:
```

```
    nat = "正常"
```

```
elif 24 <= bmi < 28:
```

```
    nat = "偏胖"
```

```
else:
```

```
    nat = "肥胖"
```

```
print("BMI 指标为:国内'{0}'".format(nat))
```

| 分类 | 国际BMI值    | 国内BMI值    |
|----|-----------|-----------|
| 偏瘦 | <18.5     | <18.5     |
| 正常 | 18.5 ~ 25 | 18.5 ~ 24 |
| 偏胖 | 25 ~ 30   | 24 ~ 28   |
| 肥胖 | ≥30       | ≥28       |

```
#CalBMIV3.py
```

```
height, weight = eval(input("请输入库里的身高(米)和体重(公斤)[逗号隔  
开]: "))
```

```
bmi = weight / pow(height, 2) print("BMI 数值为:  
{:.2f}".format(bmi)) who, nat = "", ""
```

```
if bmi < 18.5:  
    who, nat = "偏瘦", "偏瘦"
```

```
elif 18.5 <= bmi < 24:  
    who, nat = "正常", "正常"
```

```
elif 24 <= bmi < 25:  
    who, nat = "正常", "偏胖"
```

```
elif 25 <= bmi < 28:  
    who, nat = "偏胖", "偏胖"
```

```
elif 28 <= bmi < 30:  
    who, nat = "偏胖", "肥胖"
```

```
else:  
    who, nat = "肥胖", "肥胖"
```

```
print("BMI 指标为:国际'{0}', 国内'{1}'".format(who, nat))
```

| 分类 | 国际BMI值    | 国内BMI值    |
|----|-----------|-----------|
| 偏瘦 | <18.5     | <18.5     |
| 正常 | 18.5 ~ 25 | 18.5 ~ 24 |
| 偏胖 | 25 ~ 30   | 24 ~ 28   |
| 肥胖 | ≥30       | ≥28       |



"**身体质量指数BMI**"举一反三



File Edit Format Run Options Window Help

```
while True:
    height, weight = eval(input("请输入身高(米)和体重(公斤): "))
    bmi = weight / pow(height, 2)
    print("BMI 数值为: {:.2f}".format(bmi))
    who, nat = "", ""
    if bmi < 18.5:
        who, nat = "偏瘦", "偏瘦"
    elif 18.5 <= bmi < 24:
        who, nat = "正常", "正常"
    elif 24 <= bmi < 25:
        who, nat = "正常", "偏胖"
    elif 25 <= bmi < 28:
        who, nat = "偏胖", "偏胖"
    elif 28 <= bmi < 30:
        who, nat = "偏胖", "肥胖"
    else:
        who, nat = "肥胖", "肥胖"
    name = input("The player is ")
    if who != nat:
        print(name + " cried and blamed that it is unfair!")
    else:
        print(name + " feeled that it is fair.")
    print("BMI 指标为:国际'{}', 国内'{}'.format(who, nat)+"\n")
```

===== RESTART: E:/liushuo/currybmi.py =====

请输入身高(米)和体重(公斤): 1.91,90.7

BMI 数值为: 24.86

The player is Curry

Curry cried and blamed that it is unfair!

BMI 指标为:国际'正常', 国内'偏胖'

请输入身高(米)和体重(公斤): 2.11,106.6

BMI 数值为: 23.94

The player is Durant

Durant feeled that it is fair.

BMI 指标为:国际'正常', 国内'正常'

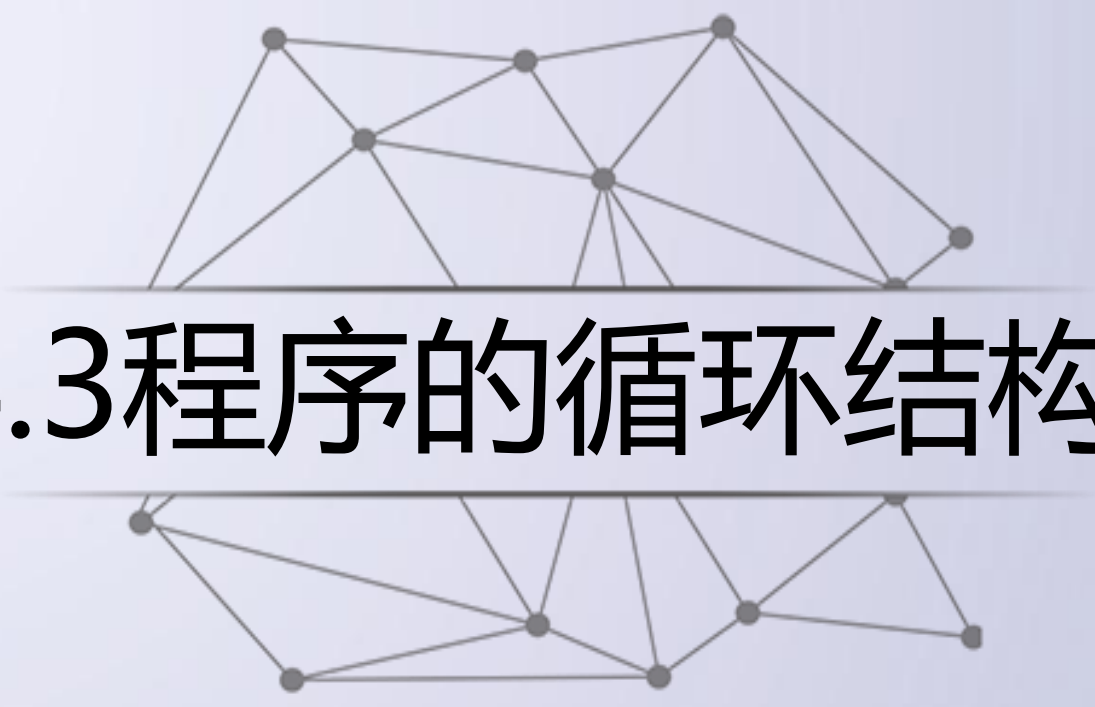
请输入身高(米)和体重(公斤):



# 举一反三

## 关注多分支条件的组合

- 多分支条件之间的覆盖是重要问题
- 程序可运行，但如果不正确，要注意多分支
- 分支结构是程序的重要框架，读程序先看分支



## 4.3程序的循环结构




# 遍历循环: for语句

遍历循环:

根据循环执行次数的确定性，循环可以分为确定次数循环和非确定次数循环。确定次数循环指循环体对循环次数有明确的定义循环次数采用遍历结构中元素个数来体现

Python通过保留字for实现“遍历循环”：

```
for <循环变量> in <遍历结构>:  
    <语句块>
```



# 遍历循环: for语句

遍历结构可以是字符串、文件、组合数据类型或range()函数:

|                    |                 |             |                 |
|--------------------|-----------------|-------------|-----------------|
| 循环N次               | 遍历文件fi的每一行      | 遍历字符串s      | 遍历列表ls          |
| for i in range(N): | for line in fi: | for c in s: | for item in ls: |
| <语句块>              | <语句块>           | <语句块>       | <语句块>           |

遍历循环还有一种扩展模式, 使用方法如下:

```
for <循环变量> in <遍历结构>:
```

```
    <语句块1>
```

```
else:
```

```
    <语句块2>
```

# 遍历循环

*for* <循环变量> *in* <遍历结构> :



<语句块>

- 由保留字for和in组成，完整遍历所有元素后结束
- 每次循环，所获得元素放入循环变量，并执行一次语句块

# 遍历循环的应用

## 计数循环(N次)

```
for i in range(N) :
```

＜语句块＞

- 遍历由range()函数产生的数字序列，产生循环

# 遍历循环的应用

## 计数循环(N次)

```
>>> for i in range(5):  
    print(i)
```

0  
1  
2  
3  
4

```
>>> for i in range(5):  
    print("Hello:",i)
```

Hello: 0  
Hello: 1  
Hello: 2  
Hello: 3  
Hello: 4

# 遍历循环的应用

计数循环(特定次)

```
for i in range(M,N,K) :
```

<语句块>

- 遍历由range()函数产生的数字序列，产生循环



# 遍历循环的应用

## 字符串遍历循环

```
for c in s:  
    <语句块>
```

- s是字符串，遍历字符串每个字符，产生循环

# 遍历循环的应用

## 字符串遍历循环

```
>>> for c in "Python123":  
    print(c, end=",")
```

P,y,t,h,o,n,1,2,3,

# 遍历循环的应用

## 列表遍历循环

```
for item in ls:  
    <语句块>
```

- ls是一个列表，遍历其每个元素，产生循环

# 遍历循环的应用

## 列表遍历循环

```
>>> for item in [123, "PY",456] :  
    print(item, end=",")
```

123,PY,456,

# 遍历循环的应用

## 文件遍历循环

```
for line in fi:  
    <语句块>
```

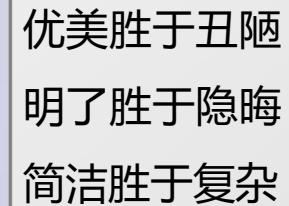
- fi是一个文件标识符，遍历其每行，产生循环

# 遍历循环的应用

## 文件遍历循环

```
>>> for line in fi :  
        print(line)
```

优美胜于丑陋  
明了胜于隐晦  
简洁胜于复杂



优美胜于丑陋  
明了胜于隐晦  
简洁胜于复杂

文件标识符

# 遍历循环

  
*for* <循环变量> *in* <遍历结构> :

<语句块>

— 计数循环(N次)      — 列表遍历循环

— 计数循环(特定次) — 文件遍历循环

— 字符串遍历循环    — .....

# 循环结构的优化



- 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。



# 循环结构的优化

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

$(x = x * 100)$   
 $(y *= 10)$   
 $(z)$   
 $x + y + z$

嵌套后z每循环一次，x y均进行一次乘法

# 循环结构的优化

## ■ 列举出全部的由 1 2 3 4组合而成的三位整数

```
File Edit Format Run Options Window Help
import time
digits = [1,2,3,4]
start = time.time()
for i in range(1000):
    result = []
    for x in digits:
        for y in digits:
            for z in digits:
                result.append(x*100+y*10+z)
print(time.time()-start)
print(result)
```

```
0.009973526000976562
[111, 112, 113, 114, 121, 122, 123, 124, 131, 132, 133, 134, 141, 142, 143, 144, 211, 212, 213, 214, 221, 222, 223, 224, 2
31, 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]
```

```
File Edit Format Run Options Window Help
import time
digits = [1,2,3,4]
start = time.time()
for i in range(1000):
    result = []
    for x in digits:
        x = x*100
        for y in digits:
            y *=10
            for z in digits:
                result.append(x+y+z)
print(time.time()-start)
print(result)
```

```
0.007978439331054688
[111, 112, 113, 114, 121, 122, 123, 124, 131, 132, 133, 134, 141, 142, 143, 144, 211, 212, 213, 214, 221, 222, 223, 224, 2
31, 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]
```

# 循环结构的优化

- 另外，在循环中应尽量引用局部变量，因为局部变量的查询和访问速度比全局变量略快，在使用模块中的方法时，可以通过将其转换为局部变量来提高运行速度。

```
>>>import time
>>>import math
```

```
>>>start = time.time() #获取当前时间
>>>for i in range(10000000):
    math.sin(i)
```

```
>>>print('Time Used:', time.time()-start) #输出所用时间
```



```
Time Used: 1.410231113433838
```

```
>>>loc_sin = math.sin
```

```
>>>start = time.time()
```

```
>>>for i in range(10000000):
```

```
    loc_sin(i)
```

```
>>>print('Time Used:', time.time()-start)
```

```
Time Used: 1.1898200511932373
```



# 无限循环: while语句

无限循环（又称**条件循环**）：

- 无限循环一直保持循环操作直到特定循环条件不被满足才结束，不需要提前知道确定循环次数。
- Python通过保留字while实现无限循环，使用方法如下：

```
while <条件>:
```

```
    <语句块>语句块
```

# 无限循环的应用

## 无限循环的条件

```
>>> a = 3
>>> while a>0:
    a= a-1
    print(a)
```

2

1

0

```
>>> a = 3
>>> while a>0 :
    a= a+1
    print(a)
```

4

5

... (CTRL + C 退出执行)

- **小练习**

关于条件循环，哪个选项的描述是错误的？

A 条件循环也叫无限循环

B 条件循环使用while语句实现

C 条件循环不需要事先确定循环次数

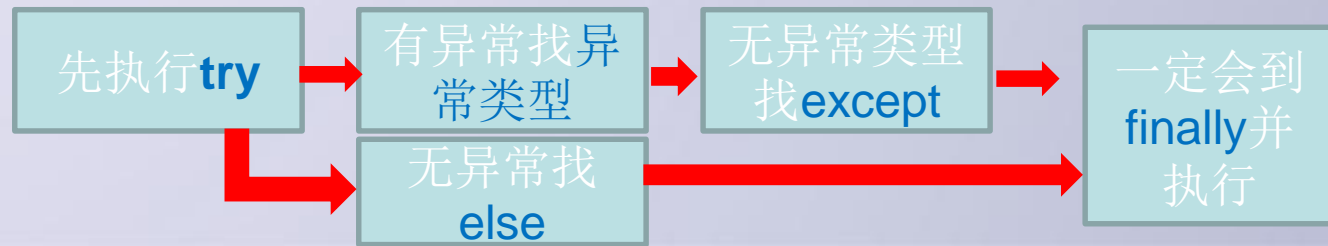
D 条件循环一直保持循环操作直到循环条件满足才结束

**D**

# 关于异常处理-续

除了try和except保留字外，异常语句还可以与else和finally保留字配合使用，语法格式如下：

|               |   |
|---------------|---|
| try:          | 正常程序在语句块1中执行。                                       |
| 语句块 1         |   |
| except异常类型1:  | 如果程序执行中 <b>发生异常</b> ，中止程序运行，跳转到所对应的异常处理块中执行。        |
| 语句块 2         |   |
| except 异常类型2: | 在“ <b>except 异常类型</b> ”语句中找对应的异常类型，如果找到的话，执行后面的语句块。 |
| 语句块 3 ...     | 如果找不到，则执行“ <b>except</b> ”后面的语句块N+2。                |
| except 异常类型N: |   |
| 语句块N+1        | 如果程序正常执行没有发生异常，则继续执行 <b>else</b> 后的语句块 N+3。         |
| except:       |   |
| 语句块N+2        | 无论异常是否发生，最后都执行 <b>finally</b> 后面语句块N+4              |
| else:         | 。   |
| 语句块N+3        |   |
| finally:      |   |
| 语句块N+4        |   |



# 关于异常处理-续

在程序运行过程中如果发生异常，Python 会输出错误消息和关于错误发生处的信息，然后终止程序。

例如：>>> short\_list = [1, 72, 3]  
>>> position = 6  
>>> short\_list[position]

Traceback (most recent call last): File "", line 1, in  
short\_list[position] IndexError: list index out of range

程序由于访问了不存在的列表元素，而发生下标越界异常

加入异常处理后

```
short_list = [1, 72, 3]
```

```
position = 6
```

```
try:
```

```
    short_list[position]
```

```
except:
```

```
    print('索引应该在 0 和', len(short_list)-1, '之间,但却是 ',position)
```

输出：索引应该在 0 和 2 之间，但却是 6



# 关于异常处理-续

## 异常处理的类型

| 异常名称               | 描述               |
|--------------------|------------------|
| SystemExit         | 解释器请求退出          |
| FloatingPointError | 浮点计算错误           |
| OverflowError      | 数值运算超出最大限制       |
| ZeroDivisionError  | 除(或取模)零 (所有数据类型) |
| KeyboardInterrupt  | 用户中断执行(通常是输入^C)  |
| ImportError        | 导入模块/对象失败        |
| IndexError         | 序列中没有此索引(index)  |
| RuntimeError       | 一般的运行时错误         |
| AttributeError     | 对象没有这个属性         |
| IOError            | 输入/输出操作失败        |
| OSError            | 操作系统错误           |
| KeyError           | 映射中没有这个键         |
| TypeError          | 对类型无效的操作         |
| ValueError         | 传入无效的参数          |

## 示例

```
===
请输入:12
请输入:0
division by zero!
executing finally clause

=====
===
请输入:23
请输入:12
result is 1.9166666666666667
executing finally clause
```

```
x = int(input("请输入:"))
y = int(input("请输入:"))
try:
    result = x / y
except ZeroDivisionError:
    print("division by zero!")
else:
    print("result is", result)
finally:
    print("executing finally clause")
```

```
short_list=[1,2,3]
```

```
while True:
```

```
    value = input('Position [q to quit]?')
```

```
    if value == 'q':
```

```
        break
```

```
    try:
```

```
        position = int(value)
```

```
        print(short_list[position])
```

```
    except IndexError:
```

```
        print('Bad index:', position)
```

```
    except:
```

```
        print('Some other error')
```

```
    else:
```

```
        print ('Thank goodness!')
```

```
    finally:
```

```
        print ('fine!')
```

```
===== RESTART: E:/liushuo/except.py ==
Position [q to quit]?wwe
Some other error
fine!
Position [q to quit]?23
Bad index: 23
fine!
Position [q to quit]?2
3
Thank goodness!
fine!
Position [q to quit]?
```



# 循环控制保留字

# 循环控制保留字


## break 和 continue

- **break**跳出并结束当前整个循环，执行循环后的语句
- **continue**结束当次循环，继续执行后续次数循环
- **break**和**continue**可以与**for**和**while**循环搭配使用

# 循环控制保留字


**break 和 continue**

```
>>> for c in "PYTHON" :  
    if c == "T" :  
        continue  
    print(c, end="")
```



PYHON

```
>>> for c in "PYTHON" :  
    if c == "T" :  
        break  
    print(c, end="")
```



PY

# 循环控制保留字

```
>>> s = "PYTHON"
>>> while s!="":
    for c in s:
        print(c, end="")
    s=s[:-1]
```

PYTHONPYTHOPYTHPYTPYP

```
>>> s = "PYTHON"
>>> while s != "" :
    for c in s :
        if c == "T":
            break
        print(c, end="")
    s=s[:-1]
```

PYPYPYPYPYP

- **break**仅跳出当前最内层循环

# 小练习

■ 下面的代码用来计算小于100的最大素数，注意break语句和else子句的用法。

```
>>> for n in range(100, 1, -1):  
    for i in range(2, n):  
        if n%i == 0:  
            break  
    else:  
        print(n)  
        break
```

97

```
>>> for n in range(100, 1, -1):  
    for i in range(2, n):  
        if n%i == 0:  
            continue  
    else:  
        print(n)  
        break
```

100  
99  
98  
97  
96  
95  
94  
93  
92  
91  
90  
89  
88  
87  
86  
85  
84  
83  
82  
81  
80  
79  
78  
77  
76  
75  
74  
73  
72  
71  
70  
69  
68  
67  
66  
65  
64  
63  
62  
61  
60  
59  
58

打印所有不能整除(2到n-1)的n  
100对应3  
99对应2  
因为continue不会跳出内层循环

For和else搭配时，  
for循环只要不是正常结束，就不会执行else。  
只要2到(n-1)之间有数字可以被n整除，就会break，否则会  
print(n)

# Continue? 在错误的道路上，**停止**就是进步

- 警惕continue可能带来的问题：

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

永不结束的死循环, Ctrl+C强行终止。



# 如何补救一下continue呢

- 这样子就不会有问题

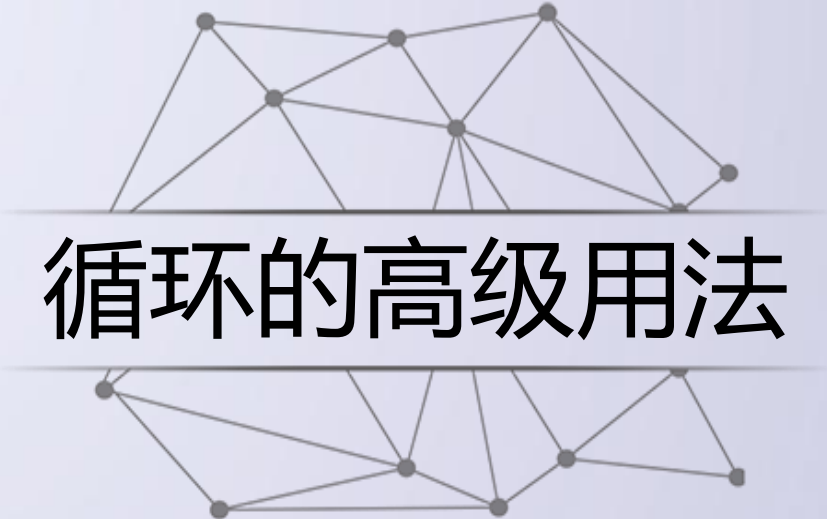
```
>>> for i in range(10):  
    if i%2==0:  
        continue  
    print(i, end=' ')
```

1 3 5 7 9

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

9

```
>>> for i in range(10):  
...     if i %2 ==0:  
...         i+=1  
...         continue  
...  
...  
>>> print(i, end=" ")
```



# 循环的高级用法

# 循环的扩展

## 循环与else

*for* <变量> *in* <遍历结构> :  
    <语句块1>

*else* :

<语句块2>

*While*:

    <语句块1>

*else* :

<语句块2>

# 循环的扩展

## 循环与else

- 当循环没有被break语句退出时，执行else语句块
- else语句块作为"正常"完成循环的奖励
- 这里else的用法与异常处理中else用法相似

# 循环的扩展


## 循环与else

```
>>> for c in "PYTHON" :  
    if c=="T" :  
        continue  
    print(c, end="")  
else:  
    print("正常退出")
```

PYHON正常退出

```
>>> for c in "PYTHON" :  
    if c== "T" :  
        break  
    print(c, end="")  
else:  
    print("正常退出")
```


PY



# 单元小结

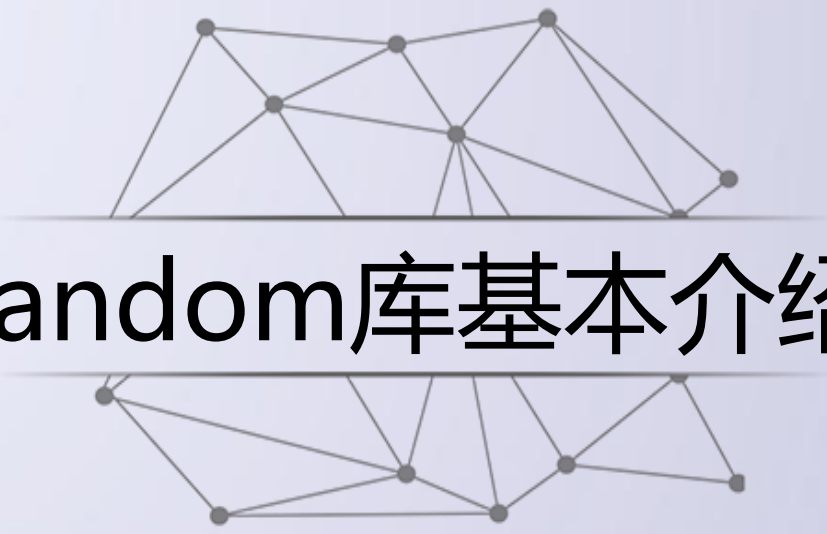
## 程序的循环结构

- *for...in* 遍历循环: 计数、字符串、列表、文件...
- *while* 无限循环
- *continue* 和 *break* 保留字: 退出当前循环层次
- 循环 *else* 的高级用法: 与 *break* 有关



## 4.4 random库的使用





# random库基本介绍

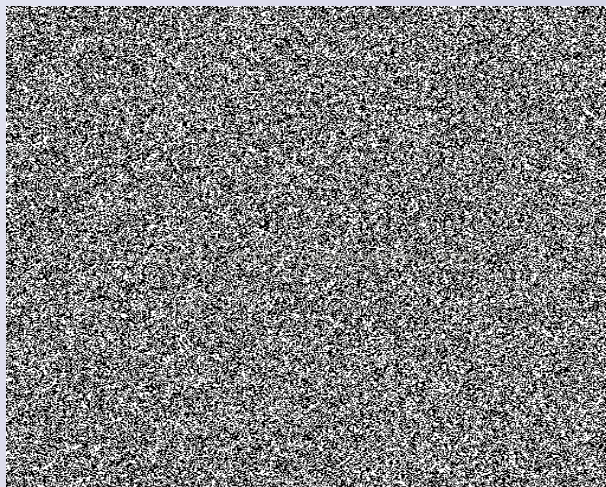
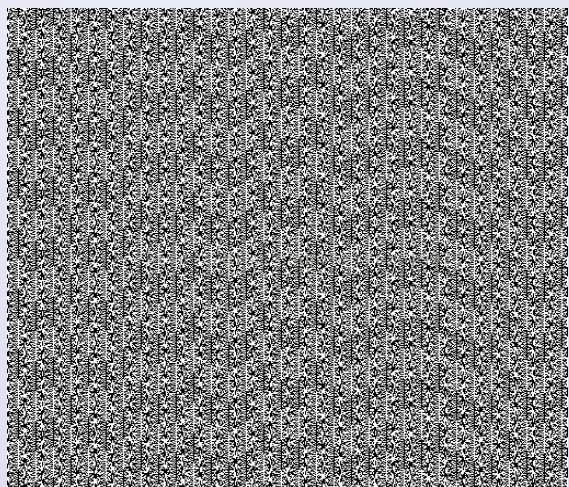
# random库概述

random库是使用随机数的Python标准库

- 伪随机数: 采用梅森旋转算法生成的(伪)随机序列中元素
- random库主要用于生成随机数
- 使用random库: `import random`

## • 真随机数和伪随机数

真正的随机数是使用物理现象/物理性随机数发生器产生的：比如掷钱币、骰子、转轮、使用电子元件的噪音、核裂变等等，它们的缺点是技术要求比较高，真随机数可谓是完美再现了生活中的真正的“随机”，也可以称为绝对的公平，结果不可见。 <https://www.random.org/>



Generator

Min:

Max:

Result:

**53**

Min: 3, Max: 100

2022-10-23 04:20:10 UTC

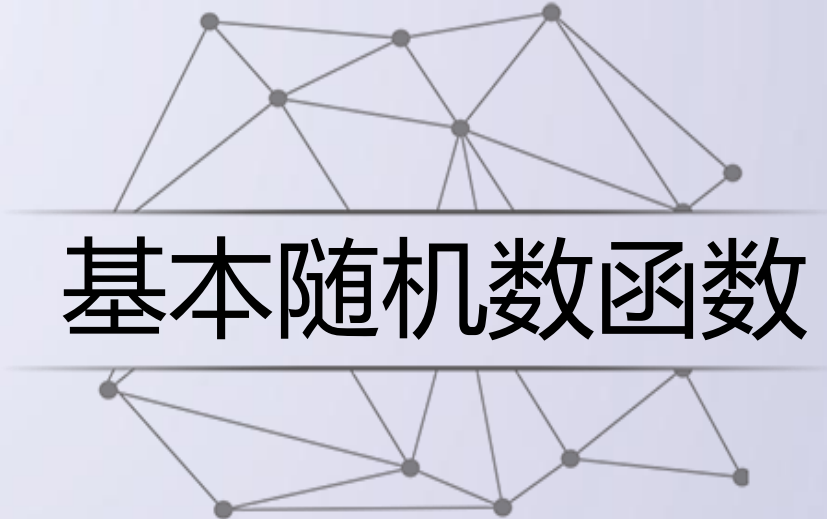
Powered by  
RANDOM.ORG

计算机中的随机函数是按照一定算法模拟产生的，其结果是确定的，是可见的。我们可以这样认为这个可预见的结果其出现的概率是100%。所以这样所产生的“随机数”并不随机，是伪随机数（依赖**种子**）。

# random库概述

**random库包括两类函数，常用共8个**

- **基本随机数函数：** `seed()`, `random()`
- **扩展随机数函数：** `randint()`, `getrandbits()`, `uniform()`,  
`randrange()`, `choice()`, `shuffle()`



# 基本随机数函数

# 基本随机数函数

随机数种子

随机数种子

10

梅森旋转算法

随机  
序  
列

0.5714025946899135

0.4288890546751146

0.5780913011344704

0.20609823213950174

0.81332125135732 随机数

0.8235888725334455

0.6534725339011758

0.16022955651881965

0.5206693596399246

0.32777281162209315

.....



# random库解析

| 函数  | 描述                               |
|---|----------------------------------|
| <code>seed(a=None)</code>                   | 初始化随机数种子，默认值为当前系统时间              |
| <code>random()</code>                       | 生成一个[0.0, 1.0)之间的随机小数            |
| <code>randint(a, b)</code>                  | 生成一个[a,b]之间的整数                   |
| <code>getrandbits(k)</code>                 | 生成一个k比特长度的随机整数                   |
| <code>randrange(start, stop[, step])</code> | 生成一个[start, stop)之间以step为步数的随机整数 |
| <code>uniform(a, b)</code>                  | 生成一个[a, b]之间的随机小数                |
| <code>choice(seq)</code>                    | 从序列类型(例如：列表)中随机返回一个元素            |
| <code>shuffle(seq)</code>                   | 将序列类型中元素随机排列，返回打乱后的序列            |
| <code>sample(pop, k)</code>                 | 从pop类型中随机选取k个元素，以列表类型返回          |

# 基本随机数函数

| 函数           | 描述  |
|--------------|---|
| seed(a=None) | 初始化给定的随机数种子, <b>默认为当前系统时间</b><br><b>&gt;&gt;&gt;random.seed(10)</b> #产生种子10对应的序列      |
| random()     | 生成一个[0.0, 1.0)之间的随机小数<br><b>&gt;&gt;&gt;random.random()</b> <b>0.5714025946899135</b> |



# 基本随机数函数

```
>>> import random
```

```
>>> random.seed(20)
```

```
>>> random.random()
```

```
0.9056396761745207
```

```
>>> random.random()
```

```
0.6862541570267026
```

```
...
```

```
>>> import random
>>> random.seed(5)
>>> print("随机数: ", random.random(), random.random(), random.random(), random.random(), random.random())
随机数: 0.6229016948897019 0.7417869892607294 0.7951935655656966 0.9424502837770503 0.7398985747399307
...
```

```
>>> import random
>>> random.seed(5)
>>> for i in range(5):
...     print("随机数: ", random.random())
...
随机数: 0.6229016948897019
随机数: 0.7417869892607294
随机数: 0.7951935655656966
随机数: 0.9424502837770503
随机数: 0.7398985747399307
```

```
>>> import random
```

```
>>> random.seed(10)
```

```
>>> random.random()
```

```
0.5714025946899135
```

```
>>> random.random()
```

```
0.4288890546751146
```

random.choice和随机密码

```
>>> import string
```

```
>>> x = string.digits + string.ascii_letters + string.punctuation
```

```
>>> x
```

```
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
>>> import random
```

```
>>> ".join([random.choice(x) for i in range(8)])
```

```
'H\\{.#=)g'
```

```
>>> ".join([random.choice(x) for i in range(8)])
```

```
'(CrZ[44M'
```

```
>>> ".join([random.choice(x) for i in range(8)])
```

```
'o_?[M>iF'
```

```
>>> ".join([random.sample(x,8)])
```

```
'9h<Ug0qr '
```

```
>>> x=list(x)
```

```
>>> random.shuffle(x)
```

```
>>> x
```

```
['.', ']', 'M', 'F', 'l', 'v', ')', '~', '?', '\\', '!', 'Z', '^', '1', 'T', 'w', 'U', 'j', ',', 'n', '8', '+', 'r', 'f', '#', 'd', 'i', '>', '(', 'H', '$', 'Q', 'L', 'N', '_-', 'b', '""', '&', 'D', 'C', 'c', 'I', 'A', '9', '|', 'u', '-', 'O', 'K', '@', 'q', '6', '}', '{', '^', '""', '/', 'E', 'S', '7', '3', ':', '5', 't', '2', 'R', 'a', 'o', 'm', '*', 'h', 'V', 'G', '0', '4', '<', 'Y', '%', '=', ':', 'P', '[', 'g', 'W', 's', 'p', 'X', 'y', 'z', 'J', 'x', 'e', 'k', 'B']
```

## • 习题

随机生成100内的10个整数 `import random`  
`random....`

随机生成0-100之间的奇数 `import random`  
`random....`

从字符串 'abcdefghij' 中随机选取4个字符  
`import random`  
`Random.sample(seq,k)`

随机选取列表['apple','pear','peach','orange','123']中的一个字符串


```
>>> import random
>>> random.random()
0.4844735762872069
>>> random.randint(0,101)
5
>>> random.randint(0,101)
94
>>> random.randint(0,101)
14
>>> (random.random())*100
69.74141705794402
>>> int((random.random())*100)
58
```

```
>>> import random
>>> random.randrange(1,100,2)
45
>>> random.randrange(1,100,2)
85
>>> random.randrange(1,100,2)
77
```

```
>>> import random
>>> random.sample('abcdefghij',4)
['j', 'e', 'g', 'b']
>>> random.sample('abcdefghij',4)
['g', 'h', 'd', 'c']
... 
```

```
>>> import random
>>> random.choice(['apple','pear','peach','orange','123'])
'pear'
>>> random.choice(['apple','pear','peach','orange','123'])
'peach'
>>> random.choice(['apple','pear','peach','orange','123'])
'apple'
```

```
>>> a=['apple','pear','peach','orange','123']
>>> a[random.randint(0,4)]
'pear'
>>> a[random.randint(0,4)]
'peach'
```



# 扩展随机数函数

# 扩展随机数函数

[0,1]

**random()**

**randint()**

**randrange()**

**getrandbits()**

**uniform()**

**choice()**

**shuffle()**



# 扩展随机数函数

| 函数                                | 描述  |
|-----------------------------------|---|
| <code>randint(a, b)</code>        | 生成一个[a, b]之间的整数<br><pre>&gt;&gt;&gt;random.randint(10, 100)</pre><br><b>64</b>              |
| <code>randrange(m, n[, k])</code> | 生成一个[m, n)之间以k为步长的随机整数<br><pre>&gt;&gt;&gt;random.randrange(10, 100, 10)</pre><br><b>80</b> |

# 扩展随机数函数

| 函数             | 描述   |
|----------------|--|
| getrandbits(k) | 生成一个k比特长的随机整数<br><b>&gt;&gt;&gt;random.getrandbits(16)    34736</b>                          |
| uniform(a, b)  | 生成一个[a, b]之间的随机小数<br><b>&gt;&gt;&gt;random.uniform(10, 100)</b><br><b>13.096321648808136</b> |

```
>>> random.getrandbits(16)
34736
...
```

```
>>> m=bin(34726)
>>> print(m)
0b1000011110100110
...
```

```
def trans(num):
    import math
    m=""
    while num > 0:
        m+=str(math.floor(num%2))
        num=num//2
    return(m[::-1])
x=eval(input("请输入十进制数: "))
print(trans(x))
```

```
>>>
```

```
=====
请输入十进制数: 34736
1000011110110000
```

# 扩展随机数函数

| 函数           | 描述  |
|--------------|---|
| choice(seq)  | 从序列seq中随机选择一个元素<br><pre>&gt;&gt;&gt;random.choice([1,2,3,4,5,6,7,8,9]) 8</pre>  |
| shuffle(seq) | 将序列seq中元素随机排列，返回打乱后的序列<br><pre>&gt;&gt;&gt;s=[1,2,3,4,5,6,7,8,9];<br/><br/>random.shuffle(s);<br/><br/>print(s) [3, 5, 8, 9, 6, 1, 2, 7, 4]</pre> |

先以列表为例





# 随机数函数的使用

## 需要掌握的能力

—能够利用随机数种子产生“确定”伪随机数

—能够产生随机整数

—能够对序列类型进行随机操作

shuffle

# 再谈while条件控制循环

## while 条件控制循环

运行规则

不断循环，直到条件表达式结果为 False

结构

while ...

while...else...

else 语句块只有在循环正常结束后使用

break 跳出循环，会跳过else语句执行

while True : ...

无限循环

需要在 执行语句块中加入 break，跳出循环

提前结束循环

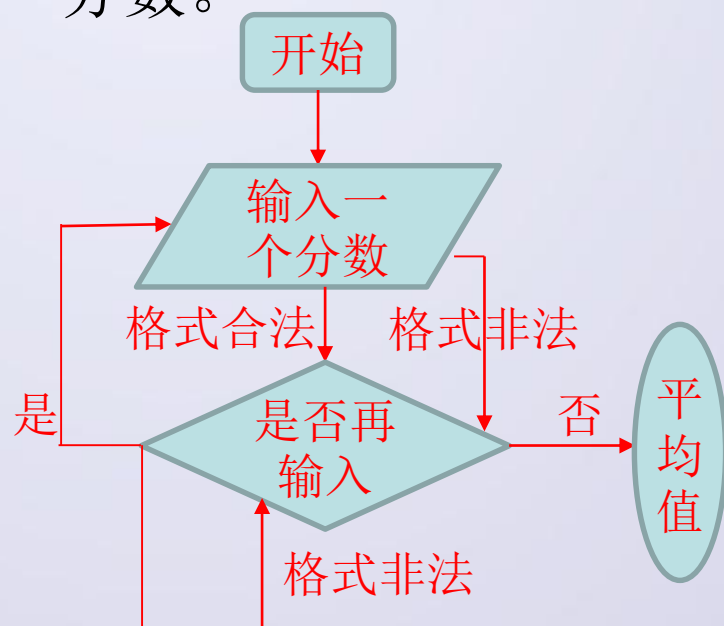
break

结束循环，不再执行循环

continue

跳过本次循环，执行下一次循环

- 小练习：用户输入若干个分数，求所有分数的平均分。每输入一个分数后询问是否继续输入下一个分数，回答“yes”就继续输入下一个分数，回答“no”就停止输入分数。



```
numbers = []                                #使用列表存放临时数据
while True:
    x = input('请输入一个成绩: ')
    try:                                     #异常处理结构
        numbers.append(float(x))
    except:
        print('不是合法成绩')
    while True:
        flag = input('继续输入吗? (yes/no) ')
        if flag.lower() not in ('yes', 'no'): #限定用户输入内容必须为yes或no
            print('只能输入yes或no')
        else:
            break
    if flag.lower() == 'no':
        break

print(sum(numbers)/len(numbers))
```

## ■ 小练习：编写程序，判断今天是今年的第几天。

```
import time
```

```
date = time.localtime() #获取当前日期时间
```

```
year, month, day = date[:3]
```

```
day_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

```
if year%400==0 or (year%4==0 and year%100!=0): #判断是否为闰年
```

```
    day_month[1] = 29
```

```
if month==1:
```

```
    print(day)
```

```
else:
```

```
    print(sum(day_month[:month-1])+day)
```

- 其中闰年判断可以直接使用calendar模块的方法。

```
>>> calendar.isleap(2022)
```

```
False
```

```
>>> calendar.isleap(2015)
```

```
False
```

- 或者使用下面的方法直接计算今天是今年的第几天

```
>>> datetime.date.today().timetuple().tm_yday
```

```
296
```

```
>>> datetime.date(2022, 10, 23).timetuple().tm_yday
```

```
296
```

- `datetime`还提供了其他功能

```
>>> now = datetime.datetime.now()
```

```
>>> now
```

```
datetime.datetime(2022, 10, 23, 15, 3, 58, 8139)
```

```
>>> now.replace(second=30)          #替换日期时间中的秒
```

```
datetime.datetime(2022, 10, 23, 15, 3, 30, 8139)
```

```
>>> now+datetime.timedelta(days=5)  #计算5天后的日期时间
```

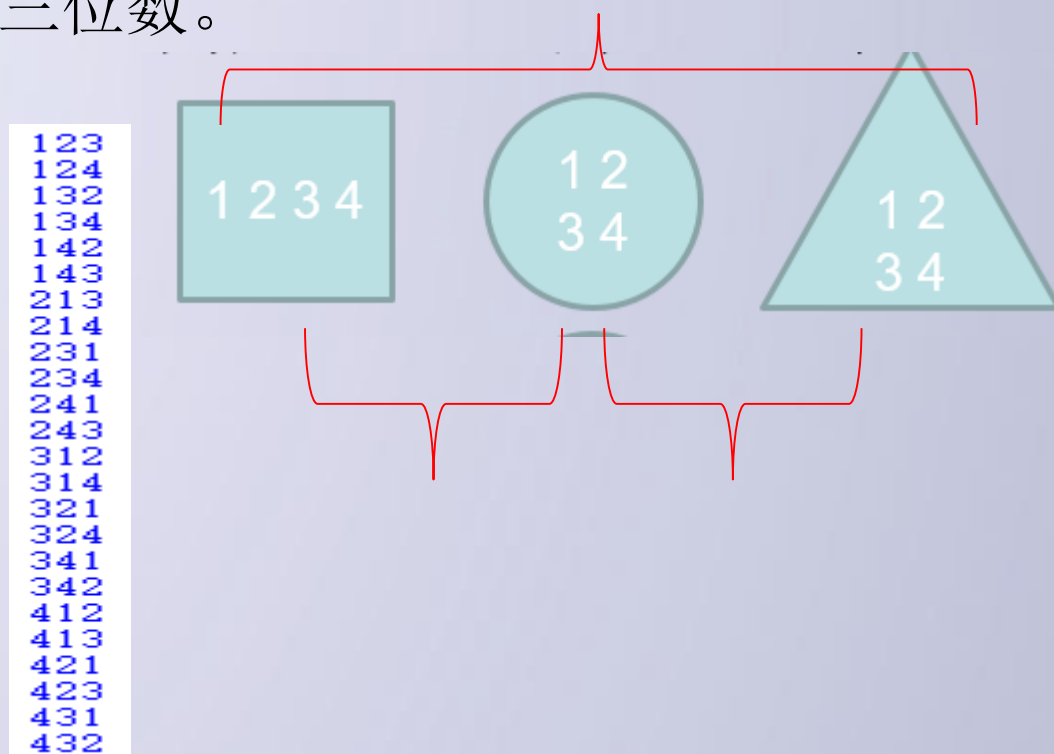
```
datetime.datetime(2022, 10, 28, 15, 3, 30, 8139)
```

```
>>> now + datetime.timedelta(weeks=-5) #计算5周前的日期时间
```

```
datetime.datetime(2022, 9, 18, 15, 3, 58, 8139)
```

- 小练习：编写程序，输出由1、2、3、4这四个数字组成的每位数都不相同的所有三位数。

```
digits = (1, 2, 3, 4)
for i in digits:
    for j in digits:
        for k in digits:
            if i!=j and j!=k and i!=k:
                print(i*100+j*10+k)
```





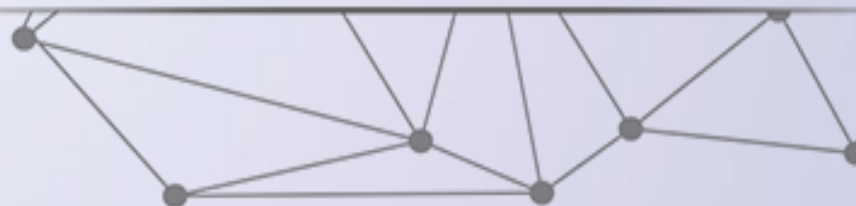
- 小练习：求1~100之间能被7整除，但不能同时被5整除的所有整数。

```
for i in range(1, 101):  
    if i % 7 == 0 and i % 5 != 0:  
        print(i)
```

```
i=0  
While i <=100:  
    i+=1  
    if i % 7 == 0 and i % 5 != 0:  
        print(i)  
        continue
```



## 4.5 $\pi$ 的计算





# $\pi$ 的计算

- $\pi$  (圆周率) 是一个无理数, 即无限不循环小数。精确求解圆周率 $\pi$ 是几何学、物理学和很多工程学科的关键。
- 对 $\pi$ 的精确求解曾经是数学历史上一一直难以解决的问题之一, 因为 $\pi$ 无法用任何精确公式表示, 在电子计算机出现以前,  $\pi$ 只能通过一些近似公式的求解得到, 直到1948年, 人类才以人工计算方式得到 $\pi$ 的808位精确小数。

$$\pi = \sum_{k=0}^{\infty} \left[ \frac{1}{16^k} \left( \frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$



# $\pi$ 的计算

随着计算机的出现，数学家找到了另类求解 $\pi$ 的另类方法：蒙特卡罗（Monte Carlo）方法，又称随机抽样或统计试验方法。当所要求解的问题是某种事件出现的概率，或者是某个随机变量的期望值时，它们可以通过某种“试验”的方法，得到这种事件出现的频率，或者这个随机变数的平均值，并用它们作为问题的解。这就是蒙特卡罗方法的基本思想。



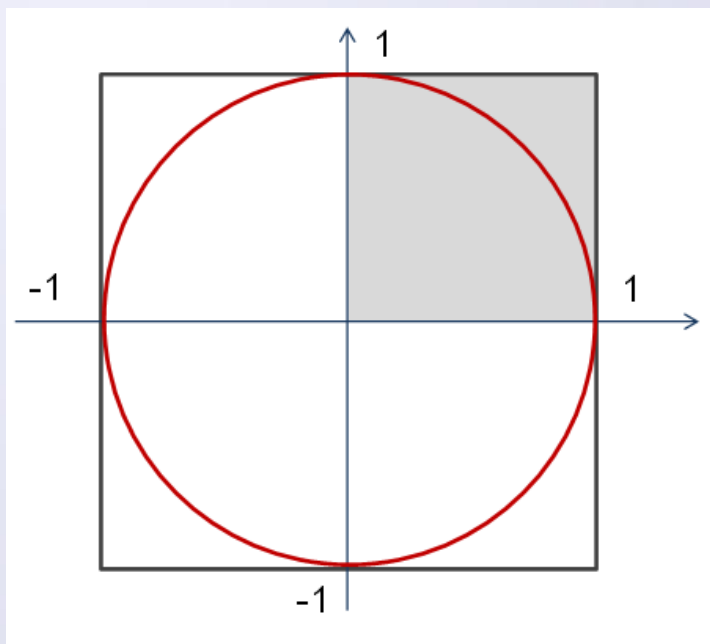
# $\pi$ 的计算

应用蒙特卡罗方法求解 $\pi$ 的基本步骤如下：

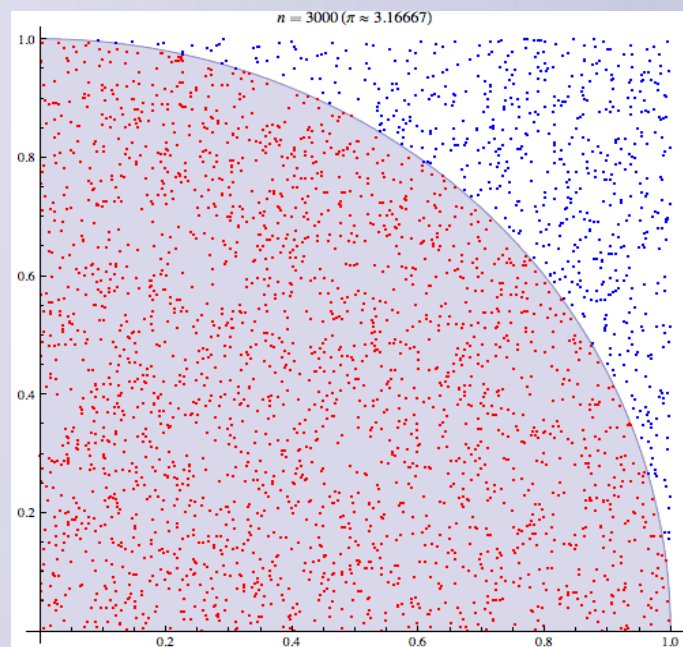
- 随机向单位正方形和圆结构，抛洒大量“飞镖”点
- 计算每个点到圆心的距离从而判断该点在圆内或者圆外
- 用圆内的点数除以总点数就是 $\pi/4$ 值。

随机点数量越大，越充分覆盖整个图形，计算得到的 $\pi$ 值越精确。实际上，这个方法的思想是利用离散点值表示图形的面积，通过面积比例来求解 $\pi$ 值。

# $\pi$ 的计算



计算 $\pi$ 使用的正方形和圆结构



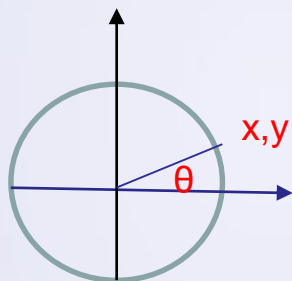
计算 $\pi$ 使用的1/4区域和抛点过程

# "圆周率的计算"实例讲解

## 圆周率的近似计算公式

$$\pi = \sum_{k=0}^{\infty} \left[ \frac{1}{16^k} \left( \frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

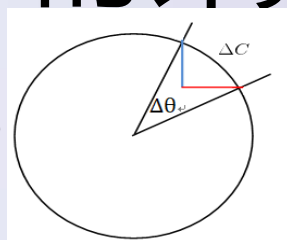
# "圆周率的计算"实例讲解



$$x^2 + y^2 = r^2$$

$$\Delta C \approx \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

$$dC \approx \sqrt{(dx)^2 + (dy)^2}$$



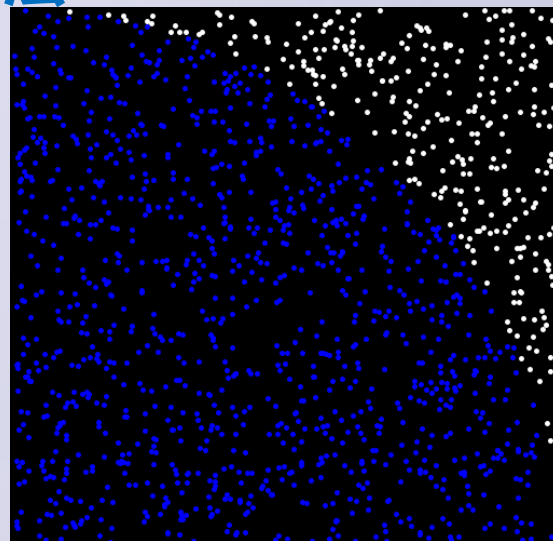
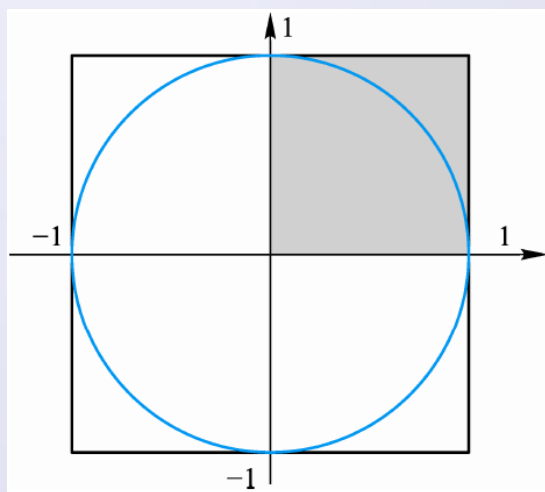
$$y = r \sin \theta \quad dy = r \cos \theta d\theta$$

$$x = r \cos \theta \quad dx = -r \sin \theta d\theta$$

$$dC = d\theta \sqrt{(-r \sin \theta)^2 + (r \cos \theta)^2}$$

$$C = \int_0^{2\pi} r d\theta = 2\pi r$$

蒙特卡罗方法

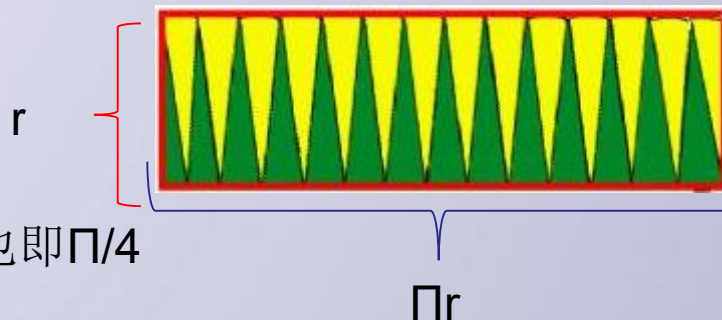


圆周长为  $2\pi r$

割圆法(把圆分为小扇形)

得面积为  $\pi r^2$

圆的面积除以正方形的面积为内部点的比值, 也即  $\pi/4$





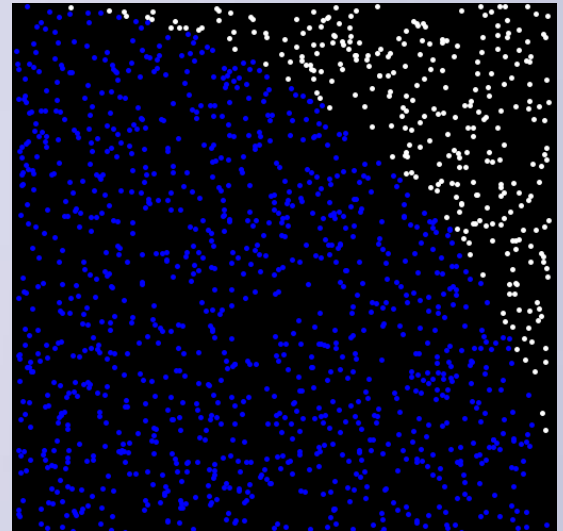
```
from random import random
from time import perf_counter

DARTS = 1000*1000

hits = 0.0
start = perf_counter()
for i in range(1, DARTS+1):
    x, y = random(), random()
    dist = pow(x ** 2 + y ** 2, 0.5)
    if dist <= 1.0:
        hits = hits + 1

pi = 4 * (hits/DARTS)
print("圆周率值是: {}".format(pi))
print("运行时间是: {:.5f}s".format(perf_counter()-start))
```

[0-1]



# π的计算

实例代码6.1

e6.1CalPi.py

```
1  #e6.1CalPi.py
2  from random import random
3  from math import sqrt
4  from time import clock
5  DARTS = 10000
6  hits = 0.0
7  clock()
8  for i in range(1, DARTS+1):
9      x, y = random(), random()
10     dist = sqrt(x ** 2 + y ** 2)
11     if dist <= 1.0:
12         hits = hits + 1
13 pi = 4 * (hits/DARTS)
14 print("Pi值是{}".format(pi))
15 print("运行时间是: {:.5}s".format(clock()))
```

然而，python3没有clock方法

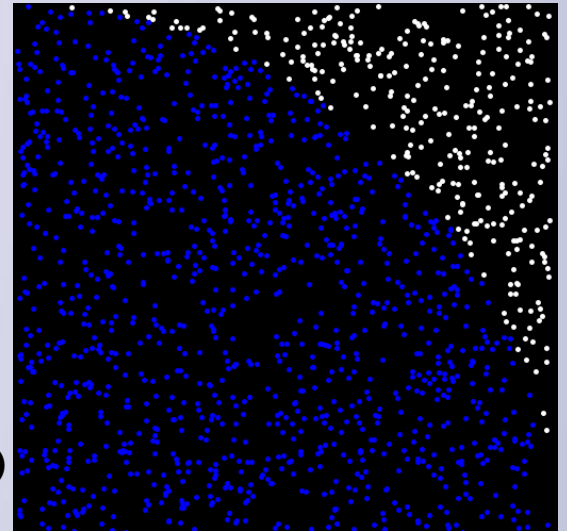
```
>>> import time
>>> time.clock
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    time.clock
AttributeError: module 'time' has no attribute 'clock'
>>> help(time.clock)
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    help(time.clock)
AttributeError: module 'time' has no attribute 'clock'
>>> help(time)
Help on built-in module time:

NAME
```

```
from random import random from time import
perf_counter DARTS = 1000*1000
hits = 0.0
start = perf_counter()
for i in range(1, DARTS+1): x, y = random(),
random()

dist = pow(x ** 2 + y ** 2, 0.5)
if dist <= 1.0: hits = hits + 1

pi = 4 * (hits/DARTS)
print("圆周率值是: {}".format(pi))
print("运行时间是: {:.5f}s".format(perf_counter()-start))
```



圆周率值是: 3.139536  
运行时间是: 0.61546s

圆周率值是: 3.1428  
运行时间是: 0.02318s

DARTS为10000

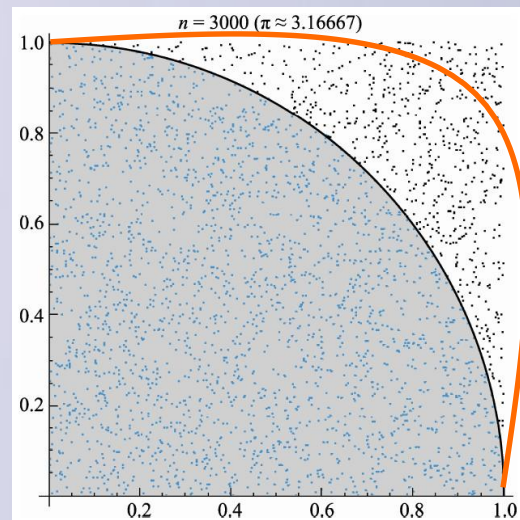
圆周率值是: 3.124  
运行时间是: 0.02065s

DARTS为1000

修改DARTS的数字, 数值也会变化

# 圆周率问题的拓展

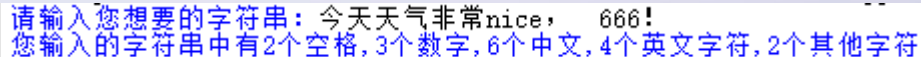
## 计算问题的扩展



- 不求解圆周率，而是某个特定图形的面积
- 在工程计算中寻找蒙特卡罗方法的应用场景

## • 作业

统计不同字符的个数，效果如下图。用户从键盘输入一行字符，编写程序，统计并且输出其中英文字符 数字 空格核其他字符的个数（对于输入的字符串i，如果i满足if  $i \geq u'\text{\u4e00}'$  and  $i \leq u'\text{\u9fa5}'$ ，则i为汉字； `i.isalpha()`对应的布尔值可以判断是不是英文字符）：



```
请输入您想要的字符串：今天天气非常nice, 666!  
您输入的字符串中有2个空格, 3个数字, 6个中文, 4个英文字符, 2个其他字符
```