



Python 程序设计

第4章 控制结构

Python基本语法元素

- 缩进、注释、命名、变量、保留字
- 数据类型、字符串、整数、浮点数、列表
- 赋值语句、分支语句、函数
- input()、print()、eval()、print()格式化



Python基本图形绘制

- 从计算机技术演进角度看待Python语言
- 海龟绘图体系及import保留字用法
- penup()、pendown()、pensize()、pencolor()
- fd()、circle()、seth()
- 循环语句：for和in、range()函数



基本数据类型

- 数据类型：整数、浮点数、复数及
- 数据类型运算操作符、运算函数
- 字符串类型：表示、索引、切片
- 字符串操作符、处理函数、处理方法、.format()格式化
- time库：time()、strptime()、strptime()、sleep()等



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

保留字



```
#TempConvert.py
```

```
TempStr = input("请输入带有符号的温度值: ")
```

```
if TempStr[-1] in ['F', 'f']:
```

```
    C = (eval(TempStr[0:-1]) - 32)/1.8
```

```
    print("转换后的温度是{:.2f}C".format(C))
```

```
elif TempStr[-1] in ['C', 'c']:
```

```
    F = 1.8*eval(TempStr[0:-1]) + 32
```

```
    print("转换后的温度是{:.2f}F".format(F))
```

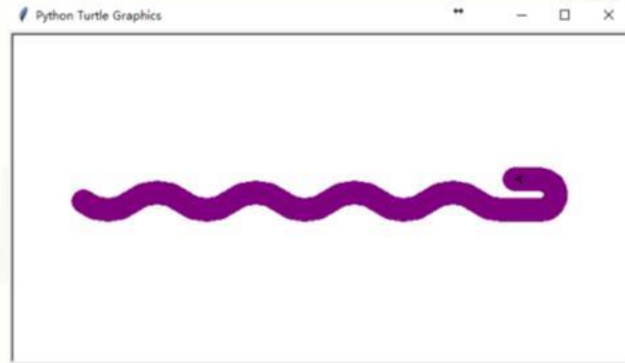
```
else:
```

```
    print("输入格式错误")
```

温度转换



```
import turtle
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for i in range(4):
    turtle.circle(40, 80)
    turtle.circle(-40, 80)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)
turtle.done()
```

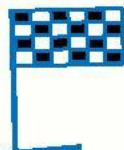
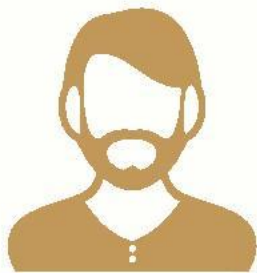


Python蟒蛇绘制



第4章 程序的控制结构

- 4.1 程序的分支结构
- 4.2 实例5: 身体质量指数BMI
- 4.3 程序的循环结构
- 4.4 模块3: random库的使用
- 4.5 实例6: 圆周率的计算

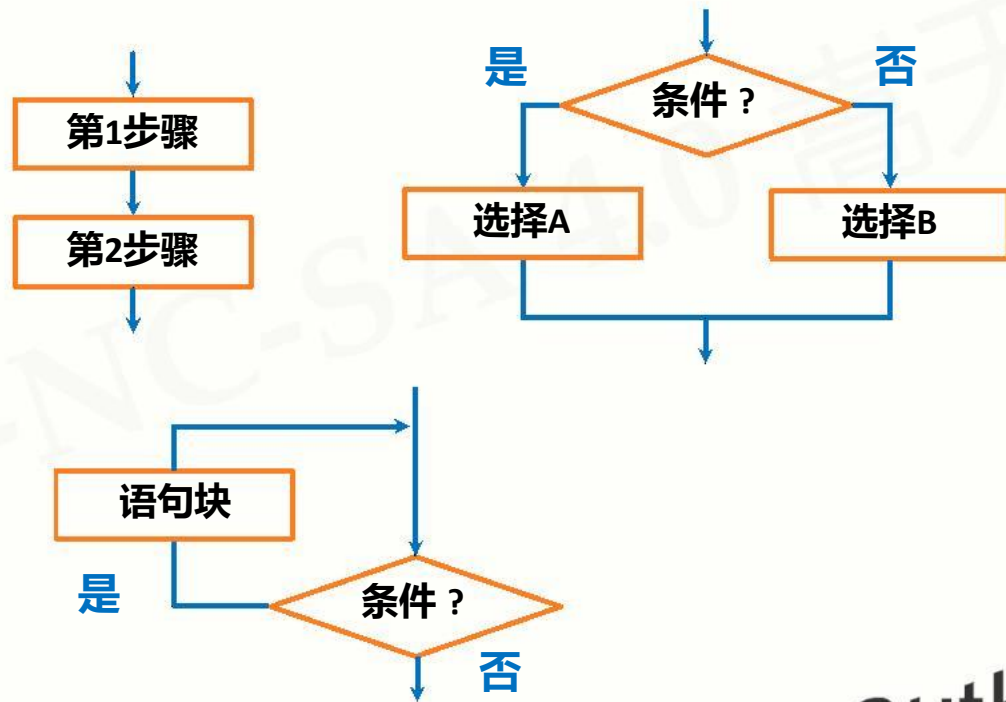


"程序的控制结构"

- 顺序结构

- 分支结构

- 循环结构



第4章 程序的控制结构

方法论

- Python程序的控制语法及结构

实践能力

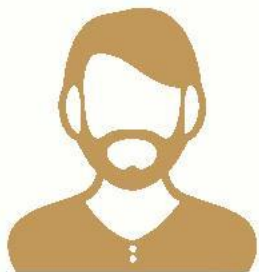
- 学会编写带有条件判断及循环的程序



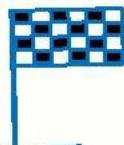
程序的分支结构



程序的分支结构



- 单分支结构
- 二分支结构
- 多分支结构
- 条件判断及组合
- 程序的异常处理

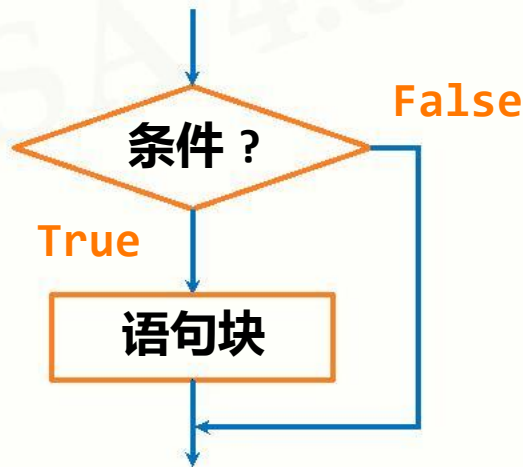


单分支结构

单分支结构

根据判断条件结果而选择不同向前路径的运行方式

if <条件> :
 <语句块>



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")
```

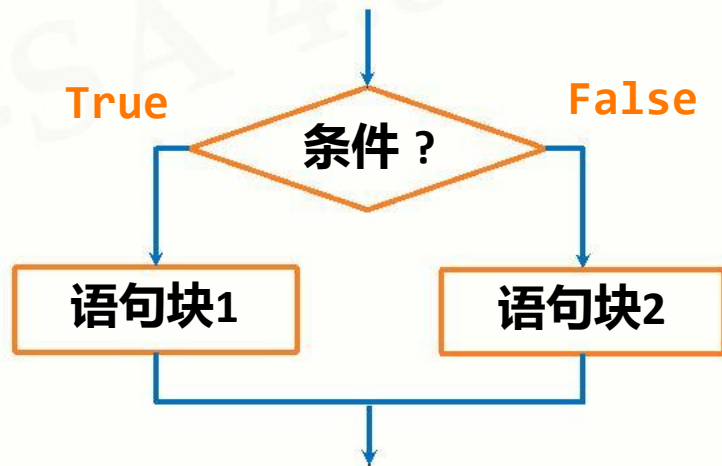



二分支结构

二分支结构

根据判断条件结果而选择不同向前路径的运行方式

if <条件> :
 <语句块1>
else :
 <语句块2>



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"))
```



多分支结构

多分支结构

if <条件1> :

<语句块1>

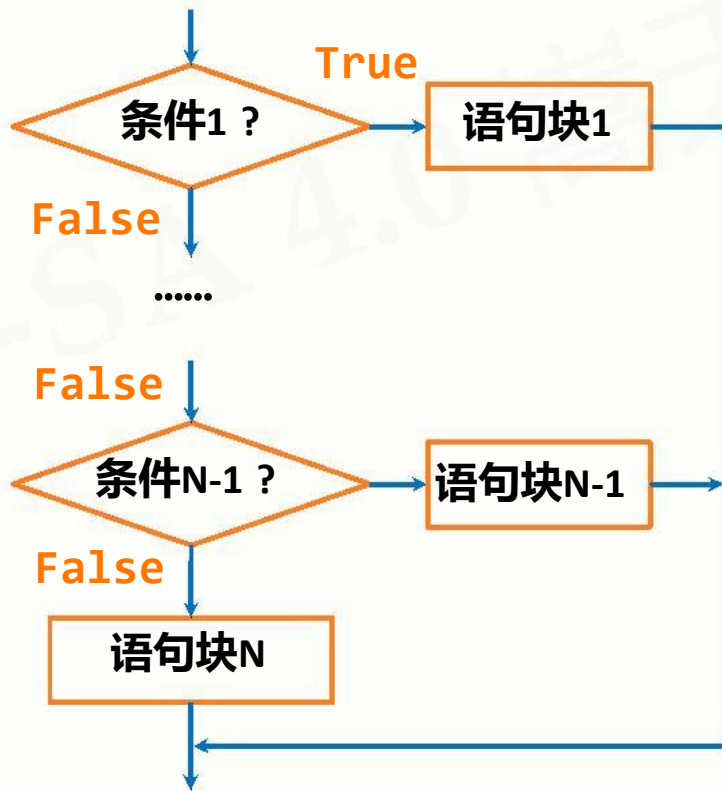
elif <条件2> :

<语句块2>

.....

else :

<语句块N>



多分支结构

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

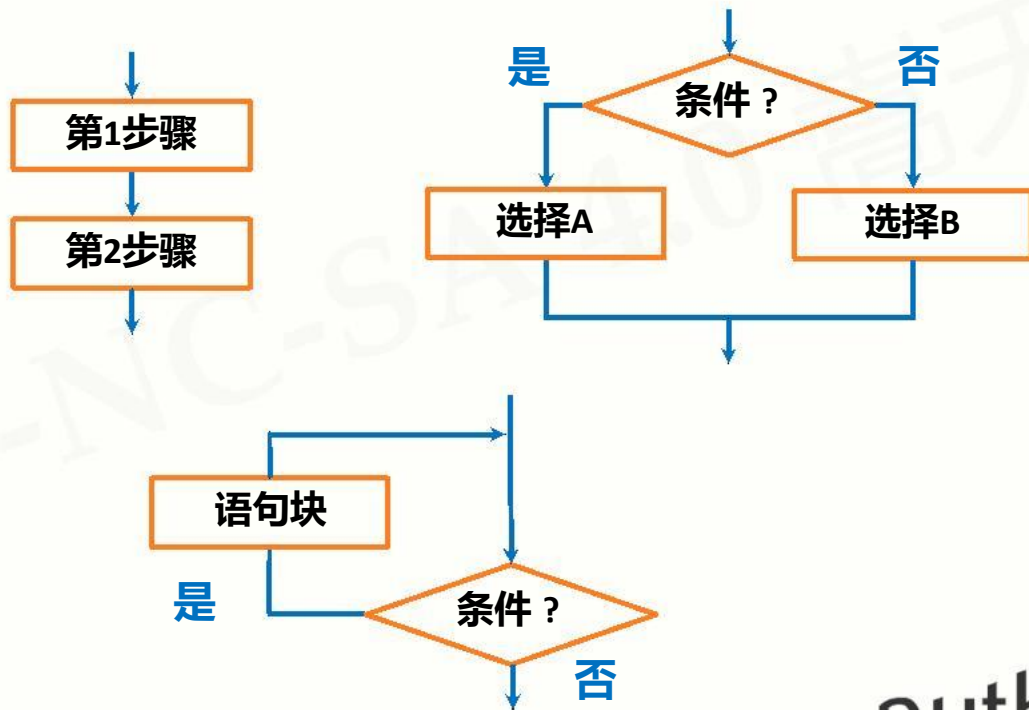
- 注意变量取值范围的覆盖

"程序的控制结构"

- 顺序结构

- 分支结构

- 循环结构





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 |

条件组合

用于条件组合的三个保留字

| 操作符及使用 | 描述 |
|----------------|---------------------|
| x and y | 两个条件x和y的逻辑 与 |
| x or y | 两个条件x和y的逻辑 或 |
| not x | 条件x的逻辑 非 |

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")
```

```
if not True:
```

```
    print("语句块2")
```

```
else :
```

```
    print("语句块1")
```



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:

```
print("输入不是整数")
```

标注异常类型后，仅响应该异常

异常类型名字等同于变量

异常处理

try :

〈语句块1〉

异常处理的高级使用

except :

〈语句块2〉

else :

〈语句块3〉

finally :

〈语句块4〉

- *finally* 对应语句块4一定执行

- *else* 对应语句块3在不发生异常时执行

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*



实例5: 身体质量指数BMI





"身体质量指数BMI"问题分析

身体质量指数BMI

BMI：对身体质量的刻画

- BMI：Body Mass Index

国际上常用的衡量人体肥胖和健康程度的重要标准，主要用于统计分析

- 定义

$$\text{BMI} = \text{体重 (kg)} / \text{身高}^2 (\text{m}^2)$$

身体质量指数BMI

BMI：对身体质量的刻画

- 实例：体重 72 kg 身高 1.75 m

BMI 值是 23.5

- 这个值是否健康呢？

身体质量指数BMI

国际：世界卫生组织 国内：国家卫生健康委员会

| 分类 | 国际BMI值 (kg/m ²) | 国内BMI值 (kg/m ²) |
|----|-----------------------------|-----------------------------|
| 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

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

```
#CalBMI.py
```

```
height, weight = eval(input("请输入身高(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"
```

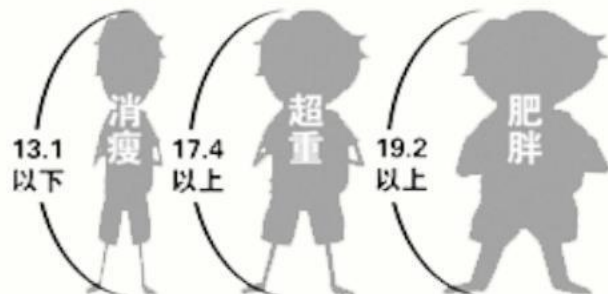
```
elif 28 <= bmi < 30:
```

```
    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))
```



举一反三

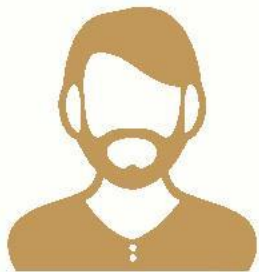
关注多分支条件的组合

- 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

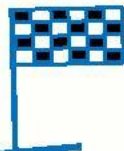
程序的循环结构



程序的循环结构



- 遍历循环
- 无限循环
- 循环控制保留字
- 循环的高级用法





遍历循环

遍历循环

遍历某个结构形成的循环运行方式

for <循环变量> *in* <遍历结构> :
 <语句块>

- 从遍历结构中逐一提取元素，放在循环变量中

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

计数循环(特定次)

```
>>> for i in range(1,6):  
    print(i)
```

1
2
3
4
5

```
>>> for i in range(1,6,2):  
    print("Hello:",i)
```

Hello: 1
Hello: 3
Hello: 5

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

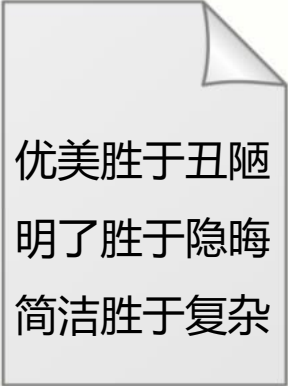
文件遍历循环

```
>>> 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

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
-

无限循环

无限循环

由条件控制的循环运行方式

while <条件> :



<语句块>

- 反复执行语句块，直到条件不满足时结束

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)



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



循环的高级用法

循环的扩展

循环与else

| | |
|------------------------------------|---------------------|
| <i>for</i> <变量> <i>in</i> <遍历结构> : | <i>while</i> <条件> : |
| <语句块1> | <语句块1> |
| <i>else</i> : | <i>else</i> : |
| <语句块2> | <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

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



模块3: 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`

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

.....

基本随机数函数

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

基本随机数函数

```
>>> import random
```

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

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

```
0.5714025946899135
```

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

```
0.4288890546751146
```

```
...
```

```
>>> import random
```

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


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

```
0.5714025946899135
```

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

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

```
0.5714025946899135
```



扩展随机数函数

扩展随机数函数

random()

```
graph LR; random[random()] --> randint[randint()]; random --> randrange[randrange()]; random --> getrandbits[getrandbits()]; random --> uniform[uniform()]; random --> choice[choice()]; random --> shuffle[shuffle()];
```

randint()

randrange()

getrandbits()

uniform()

choice()

shuffle()

扩展随机数函数

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

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 <code>>>>random.getrandbits(16)</code> <code>37885</code> |
| uniform(a, b) | 生成一个[a, b]之间的随机小数 <code>>>>random.uniform(10, 100)</code> <code>13.096321648808136</code> |

扩展随机数函数

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

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

实例6: 圆周率的计算





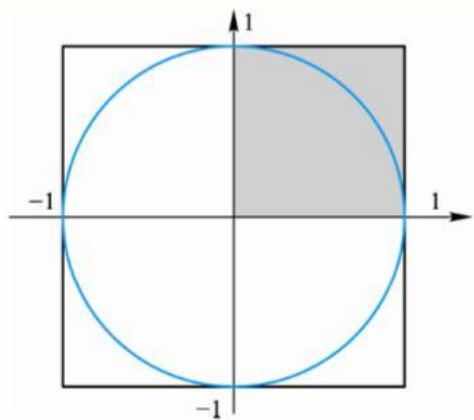
"圆周率的计算"问题分析

"圆周率的计算"问题分析

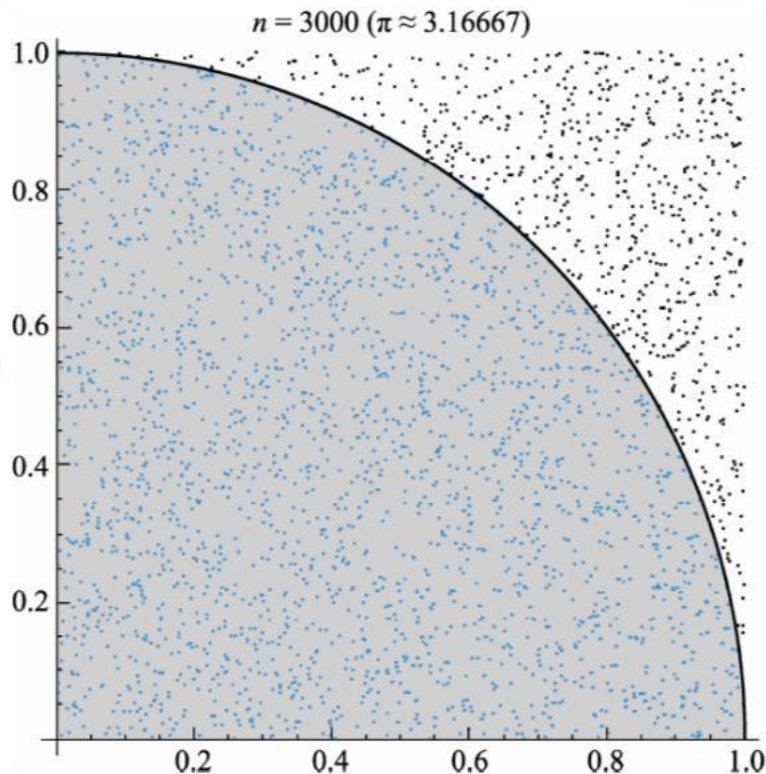
圆周率的近似计算公式

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

"圆周率的计算"问题分析



蒙特卡罗方法





"圆周率的计算"实例讲解

"圆周率的计算"实例讲解

圆周率的近似计算公式

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

```
#CalPiV1.py
```

```
pi = 0
```

```
N = 100
```

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

```
    pi += 1/pow(16,k)*(\
        4/(8*k+1) - 2/(8*k+4) - \
        1/(8*k+5) - 1/(8*k+6))
```

```
print("圆周率值是: {}".format(pi))
```

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

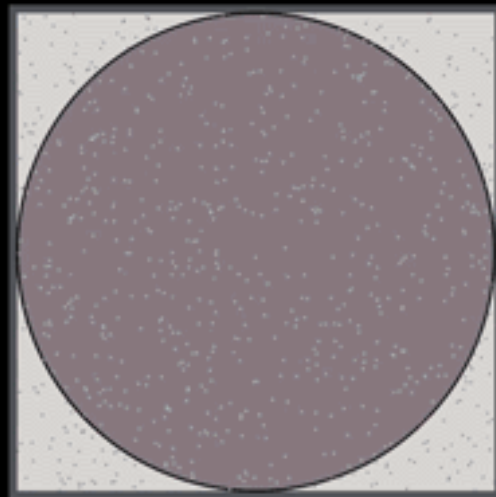
圆周率值是: 3.141592653589793

"圆周率的计算"实例讲解

蒙特卡罗方法

$$\pi \approx 4 \times \frac{\text{points within a circle}}{\text{total number of points}}$$

$$3.025 = 4 \left(\frac{605}{800} \right)$$



```
#CalPiV2.py
```

```
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)
```

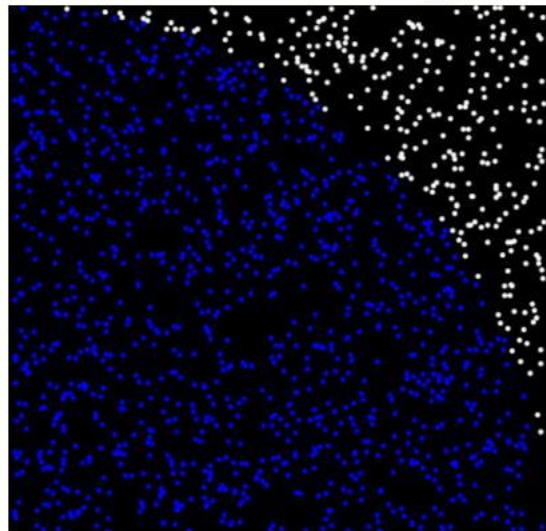
```
    if dist <= 1.0:
```

```
        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))
```





"圆周率的计算"举一反三

#CalPiV2.py

```
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)
```

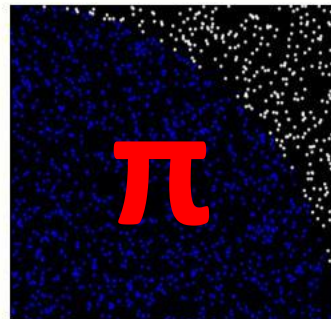
```
    if dist <= 1.0:
```

```
        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))
```



举一反三

理解方法思维

- **数学思维：找到公式，利用公式求解**
- **计算思维：抽象一种过程，用计算机自动化求解**
- **谁更准确？ (不好说...)**

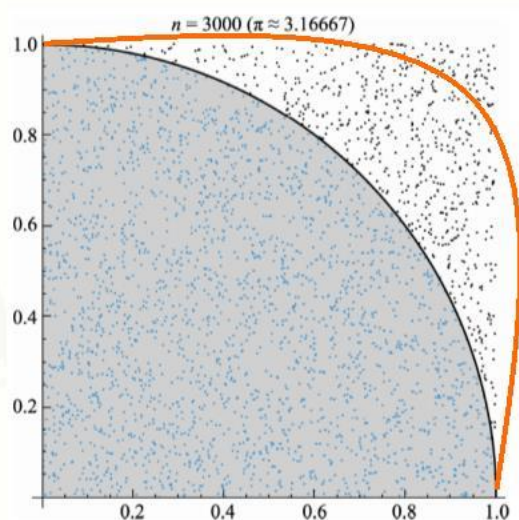
举一反三

程序运行时间分析

- 使用time库的计时方法获得程序运行时间
- 改变撒点数量，理解程序运行时间的分布
- 初步掌握简单的程序性能分析方法

举一反三

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