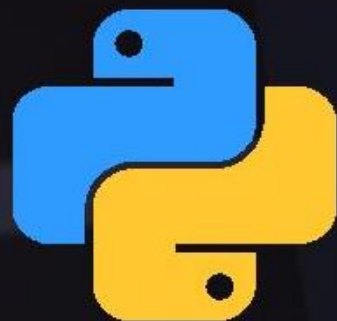


# Python程序语言入门与应用



# python

Life is short, use Python  
人生苦短，我用Python



# 课程回顾



## Python程序语法元素分析

### turtle库绘制蟒蛇

-  Python标准库的导入与使用 (import)

-  模块化编程和面向对象


## Python语言的基本数据类型


### 数字

-  3种类型

-  数值运算操作符、数值运算函数、数值类型转换函数

### 字符串

-  字符串的表示

-  字符串操作符、处理函数和方法

-  字符串format()格式化方法



# 上周练习题

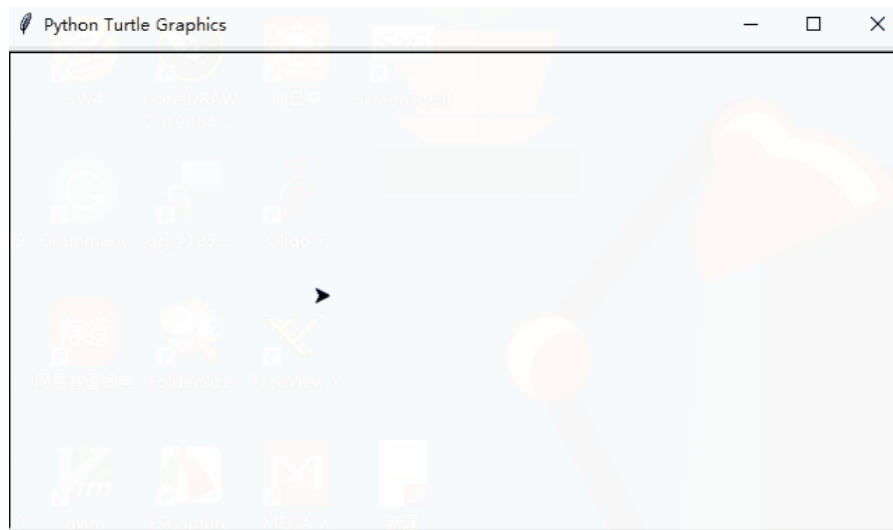


## 2.2 绘制彩色蟒蛇

蟒蛇每一小段使用不同绘制颜色。



```
# 2.2DrawColorPython.py
import turtle
def drawSnake(radius, angle, length):
    turtle.seth(-angle/2)
    for i in range(length):
        turtle.pencolor("red")
        turtle.circle(radius, angle)
        turtle.pencolor("green")
        turtle.circle(-radius, angle)
        turtle.pencolor("blue")
        turtle.circle(radius, angle/2)
        turtle.fd(radius)
        turtle.circle(radius*0.4, 180)
        turtle.fd(radius * 2/3)
turtle.setup(650, 350, 200, 200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
drawSnake(40, 80, 4)
turtle.hideturtle()
turtle.done()
```



运行示例

程序代码



# 上周练习题

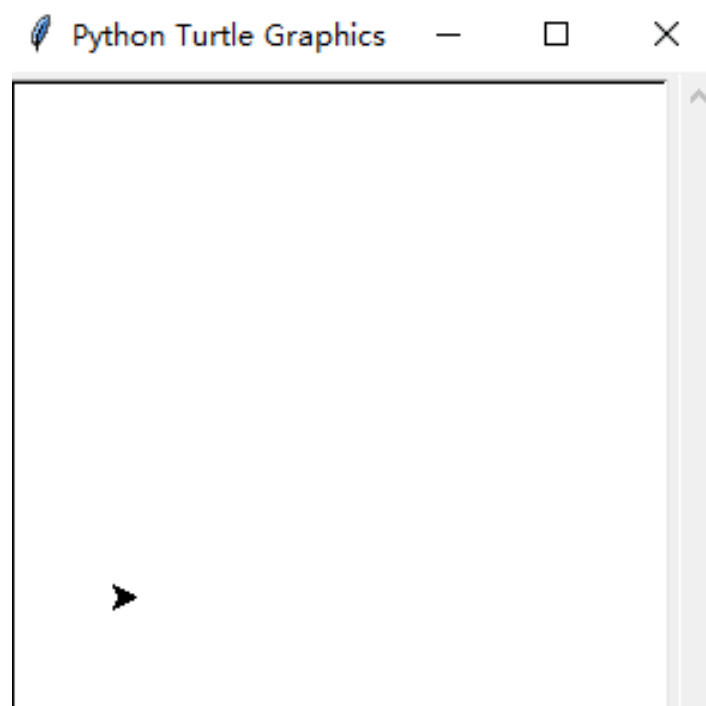


## 2.3 绘制等边三角形

使用turtle库中turtle.fd()与turtle.seth()绘制等边三角形

```
# 2.3DrawEquilateralTriangle.py
import turtle

side_length = 200
turtle.setup(300, 450, 200, 200)
turtle.penup()
turtle.fd(-side_length/2)
turtle.pendown()
turtle.pensize(2)
#turtle.pencolor("black")
for i in range(3):
    turtle.fd(side_length)
    turtle.seth((i + 1) * 120)
turtle.hideturtle()
turtle.done()
```



程序代码

运行示例

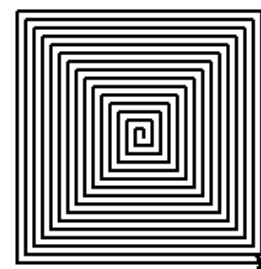


# 上周练习题

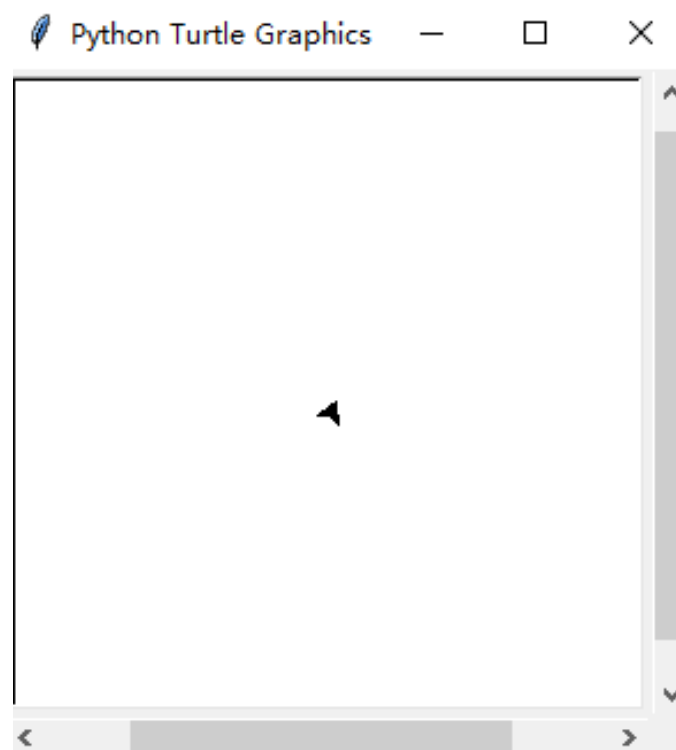


## 2.4 绘制正方形螺线

利用turtle库绘制一个正方形螺旋线。



```
# 2.4DrawSquareHelix.py
import turtle
min_side_length = 5
step_length = 5
helix_num = 30
turtle.setup(300, 300, 200, 200)
turtle.pendown()
turtle.pensize(2)
for i in range(helix_num):
    turtle.seth(90 + (i % 2) * 180)
    turtle.fd(min_side_length)
    turtle.seth(180 + (i % 2) * 180)
    turtle.fd(min_side_length)
    min_side_length += step_length
turtle.hideturtle()
turtle.done()
```



程序代码

运行示例

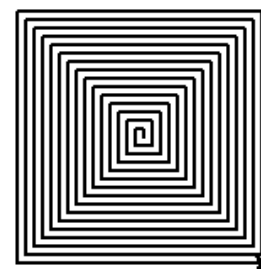


# 上周练习题



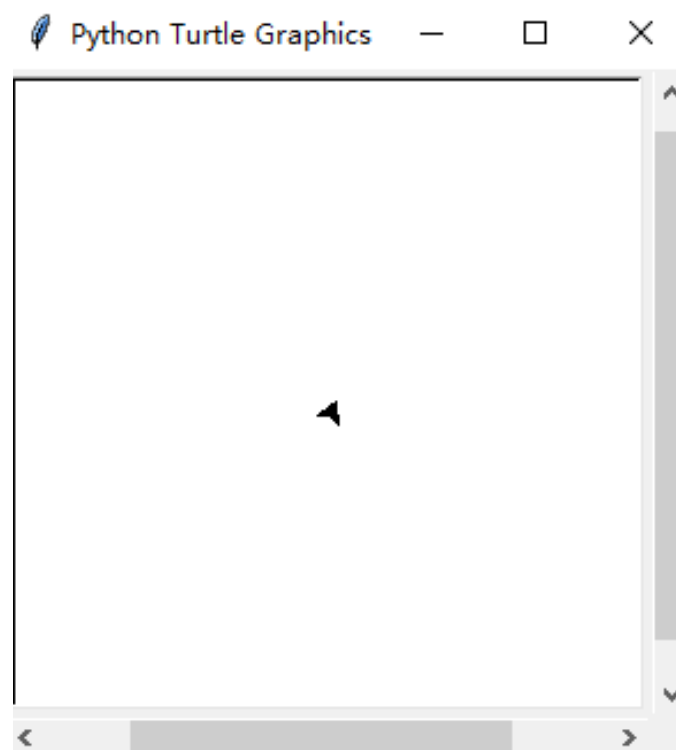
## 2.4 绘制正方形螺线

利用turtle库绘制一个正方形螺旋线。



```
# 2.4DrawSquareHelix.v2.py
import turtle
min_side_length = 5
step_length = 5
helix_num = 30
turtle.setup(300, 300, 200, 200)
turtle.pendown()
turtle.pensize(2)
for i in range(helix_num):
    for j in range(2):
        turtle.left(90)
        turtle.fd(min_side_length)
    min_side_length += step_length
turtle.hideturtle()
turtle.done()
```

程序代码



运行示例



# 上周练习题



## 3.1 重量计算

月球上物体重量为在地球上的16.5%，假如某人在地球上每年增长0.5kg，编写程序输出该人未来10年在地球上和

```
# 3.1EarthMoonWeight.py
start_weight = 50
moon_rate = 0.165
growth = 0.5
years = 10
for i in range(1, years+1):
    weight_on_earth = start_weight +
    growth * i
    weight_on_moon = weight_on_earth
    * moon_rate
    print("第{:02d}年，在地球和月球上
    的体重分别为: {:.3f}kg、{:.3f}kg"\
        .format(i, weight_on_earth,
        weight_on_moon))
```

```
gcj@DESKTOP-4F9IU9J: /mnt/f/Python2019
(base) gcj@DESKTOP-4F9IU9J: /mnt/f/Python2019$ python3 3.1EarthMoonWeight.py
第01年，在地球和月球上的体重分别为: 50.500kg、8.332kg
第02年，在地球和月球上的体重分别为: 51.000kg、8.415kg
第03年，在地球和月球上的体重分别为: 51.500kg、8.498kg
第04年，在地球和月球上的体重分别为: 52.000kg、8.580kg
第05年，在地球和月球上的体重分别为: 52.500kg、8.662kg
第06年，在地球和月球上的体重分别为: 53.000kg、8.745kg
第07年，在地球和月球上的体重分别为: 53.500kg、8.828kg
第08年，在地球和月球上的体重分别为: 54.000kg、8.910kg
第09年，在地球和月球上的体重分别为: 54.500kg、8.992kg
第10年，在地球和月球上的体重分别为: 55.000kg、9.075kg
(base) gcj@DESKTOP-4F9IU9J: /mnt/f/Python2019$
```

运行示例

程序代码



# 上周练习题



## 3.2 回文数判断

如果一个自然数 $n$ （如1234321）的各位数字反向排列所得数字与 $n$ 相等，则 $n$ 为回文数。编写程序，从键盘输入一个5位数字，判断该数字是不是回文数

```
# 3.2PalindromeNumber.py
def isPalindrome(num):
    num_reverse = num[::-1]
    if num == num_reverse:
        return True

num = input("请输入一个自然数N: ")
while num.upper() != "N":
    num_isPal = isPalindrome(num)
    if num_isPal:
        print("{}是一个回文数。".format(num))
    else:
        print("{}不是一个回文数。".format(num))
    num = input("请输入一个自然数N: ")
```

程序代码

```
gcj@DESKTOP-4F9IU9J: /mnt/f/Python2019
(base) gcj@DESKTOP-4F9IU9J: /mnt/f/Python2019$ python3 3.2PalindromeNumber.py
请输入一个自然数N: 10
10不是一个回文数。
请输入一个自然数N: 121
121是一个回文数。
请输入一个自然数N: 12345678987654321
12345678987654321是一个回文数。
请输入一个自然数N: N
(base) gcj@DESKTOP-4F9IU9J: /mnt/f/Python2019$
```

运行示例





# 上周练习题



## 3.3 输出田字格

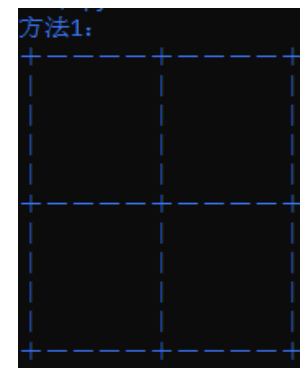
使用print函数打印出田字格。

```
# 方法 1
print("+ - - - + - - - + ")
print("|           |           |")
print("|           |           |")
print("|           |           |")
print("|           |           |")
print("+ - - - + - - - + ")
print("|           |           |")
print("|           |           |")
print("|           |           |")
print("+ - - - + - - - + ")
```

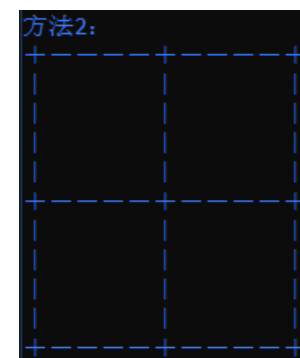
程序代码

```
# 方法 2
matts_width = 11      # 奇数
a, b, c, d = "+", "-", "|", " "
e = matts_width // 2
for i in range(matts_width):
    if i in [0, e, matts_width-1]:
        print("{}{}{}{}{}{}".format(a, b*(e-1), a, b*(e-1), a))
    else:
        print("{}{}{}{}{}{}".format(c, d*(e-1), c, d*(e-1), c))
```

方法1:



方法2:



运行示例



# 上周练习题



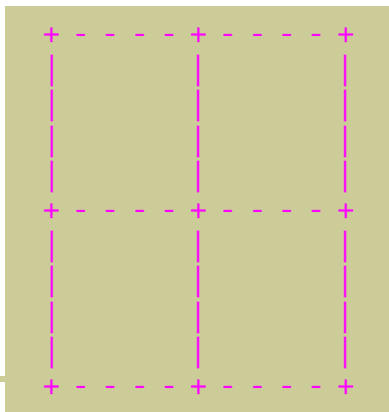
## 3.3 输出田字格

使用print函数打印出田字格。

"""打印田字格（换行间距和空格距离不知道怎么调整，还有横线用的减号，和竖线没法子一样大小）"""

```
for i in range(2):
    print("+ - - - - + - - - - +\n")
    for i in range(4):
        print("|           |           |\n")
    print("+ - - - - + - - - - +\n")
```

乔同学作业代码



全角字符: "+", "-", "|", " "  
半角字符: "+", "-", "|", " "

全角模式: 输入一个字符占用2个字符;  
半角模式: 输入一个字符占用1个字符。

h e l l o , w o r l d |

 中 ● °, 键盘 用户 工具  
<http://blog.csdn.net>

hello, world|

 中 ● °, 键盘 用户 工具  
<http://blog.csdn.net>



新乡医学院

# Python程序语言入门与应用

## 深入Python

## 第四章 Python程序控制结构



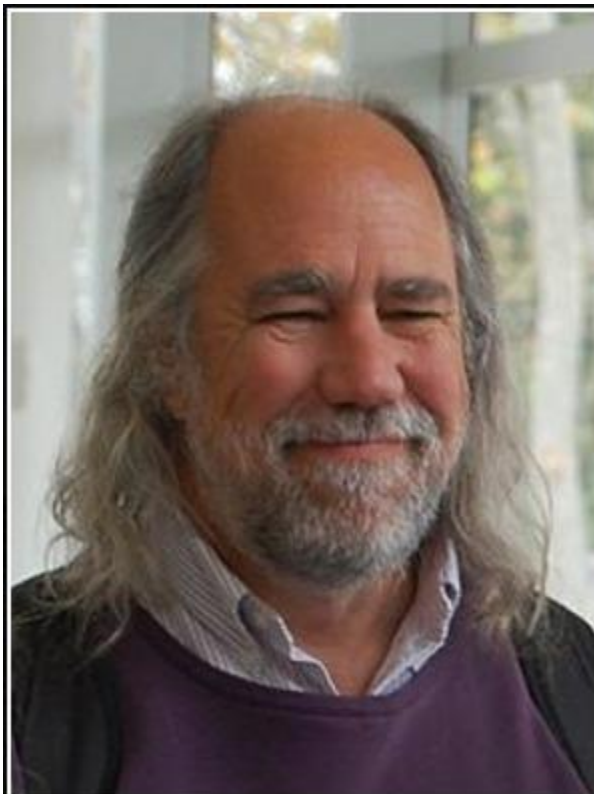
郭长江

changjiangguo@xxmu.edu.cn

生命科学技术学院

新乡医学院





The function of good software is to  
make the complex appear to be  
simple.

— Grady Booch —

AZ QUOTES

**好的软件的作用是让复杂的东西看起来简单。**

——格雷迪·布奇（UML和Booch方法创始人）




# 学习目标



## 基本要求

### 掌握

-  Python语言的分支结构与循环结构

### 理解

-  随机库的使用方法

### 了解

-  程序的异常处理及用法



# 本课概要



## 🔗 第4章 Python程序控制结构

🔗 4.1 程序的基本结构

🔗 4.2 程序的分支结构

🔗 4.3 实例：身体质量指数BMI

🔗 4.4 程序的循环结构

🔗 4.5 random库的使用

🔗 4.6  $\pi$ 值的计算

🔗 4.7 程序异常处理



## 4.1 程序的基本结构



# 4.1 程序的基本结构

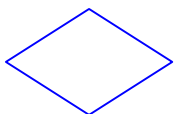


## 程序流程图

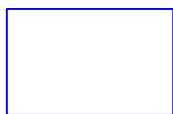
### 7种基本元素



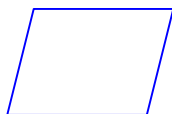
起止框



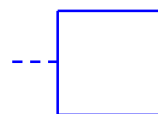
判断框



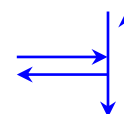
处理框



输入/输出框



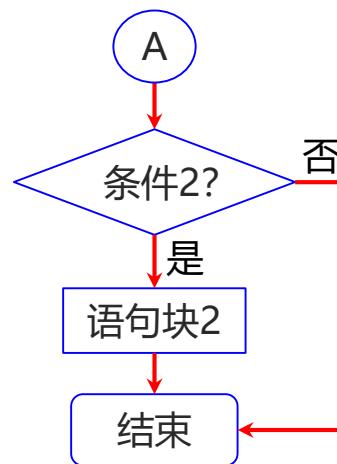
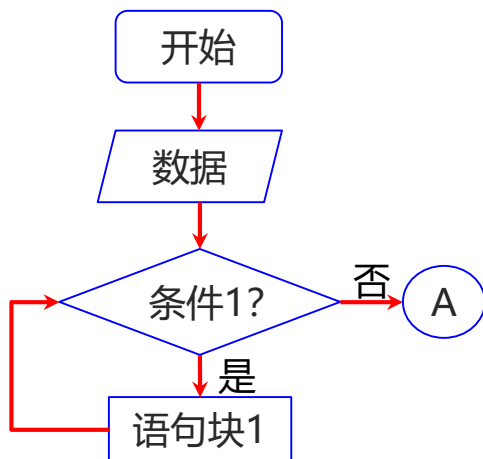
注释框



流向线



连接点



程序流程图示例：由连接点A连接的一个程序





# 4.1 程序的基本结构



## 程序的的基本结构

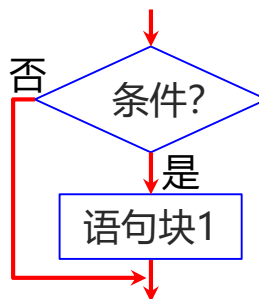
顺序结构

分支结构

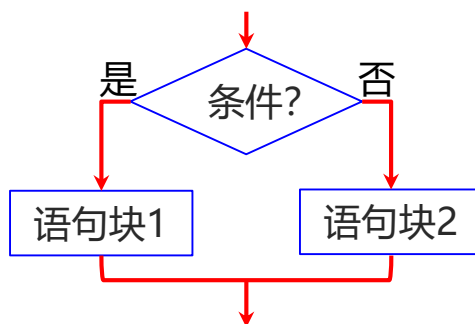
循环结构



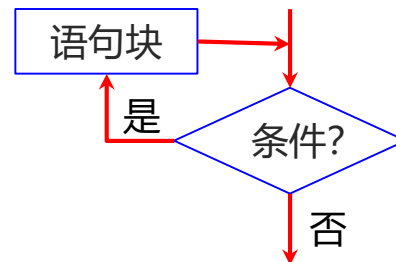
顺序结构



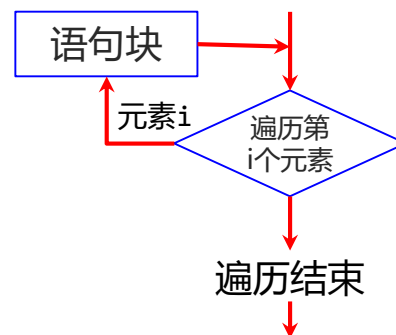
单分支结构



二分支结构



条件循环结构



遍历循环结构

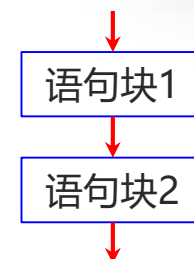


# 4.1 程序的基本结构



## 程序的基本结构实例

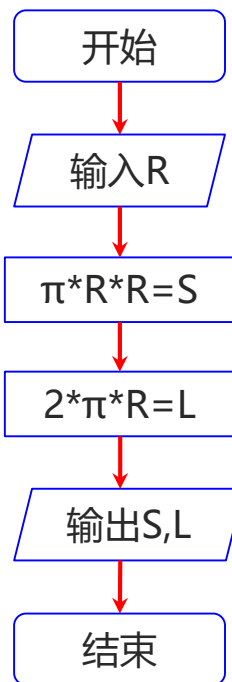
### 圆面积和周长的计算



顺序结构

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



流程图描述

```
R = eval(input("请输入圆半径: "))
S = 3.14 * R * R
L = 2 * 3.14 * R
print("圆的面积和周长: ", S, L)
```

Python代码描述



# 4.1 程序的基本结构

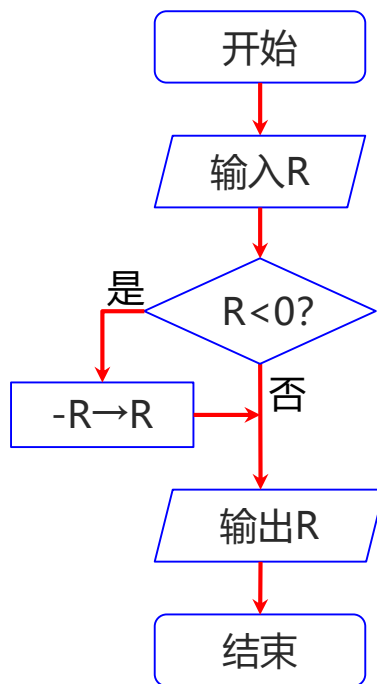


## 程序的的基本结构实例

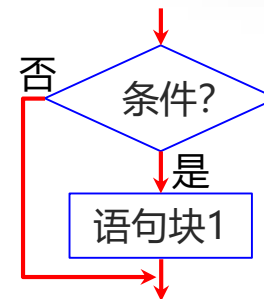
### 实数绝对值的计算

输入：实数R  
处理：  
 $|R| = R (R \geq 0)$  or  $-R (R < 0)$   
输出：|R|

问题IPO描述



流程图描述



单分支结构

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

Python代码描述



# 4.1 程序的基本结构

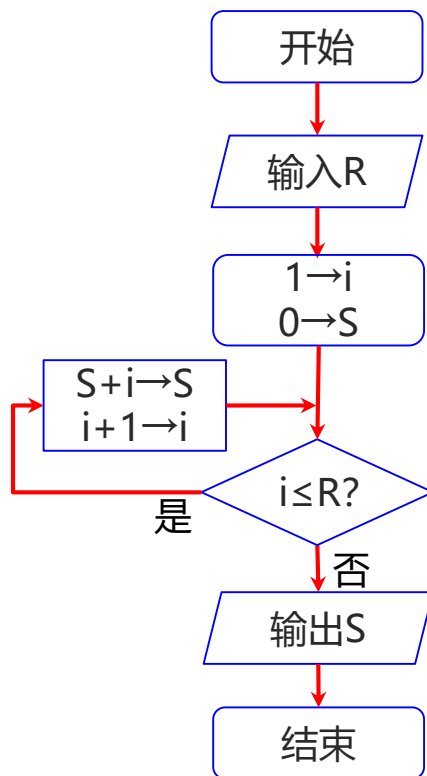


## 程序的的基本结构实例

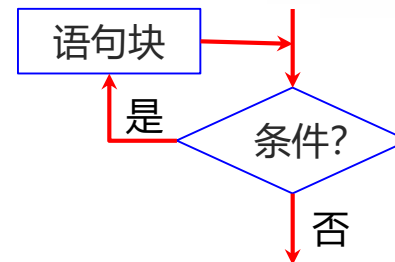
### 整数累加的计算

输入：正整数R  
处理：  
 $S=1+2+3+\dots+R$   
输出：S

问题IPO描述



流程图描述



条件循环结构

```
R = eval(input("请输入正整数: "))
i, S = 1, 0
while i <= R:
    S = S + i
    i = i + 1
print("累加求和: ", S)
```

Python代码描述



## 4.2 程序的分支结构



## 4.2 程序的分支结构



### 单分支结构：if语句

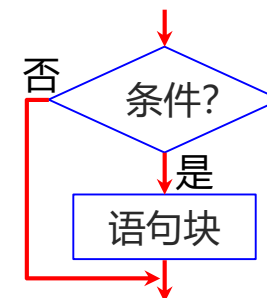
#### 语法格式

```
if <条件>:  
    <语句块>
```

#### 条件判断

产生True或False的语句或函数

最常见方式：关系操作符



单分支结构

操作符	数学符号	操作符含义	操作符	数学符号	操作符含义
<	<	小于	<=	≤	小于或等于
>	>	大于	>=	≥	大于或等于
==	=	等于	!=	≠	不等于



## 4.2 程序的分支结构



### 单分支结构

#### 关系操作符

```
>>> 1 > 2
False
>>> 1 < 2
True
>>> 1 < 1.0
False
>>> 1 == 1.0
True
>>> 1 <= 1
True
>>> 1 <= 2
True
>>> 1 >= 2
False
>>> 1 >= 1
True
```

```
>>> 1 < 2 < 3
True
>>> 1 < 2 > 1
True
>>> 1 < 2 and 2 < 3
True
>>> 1 < 2 < 3 < 4 < 5 < 6
True
>>> 'python' == 'python'
True
>>> 'Python' < 'python'
True
```



## 4.2 程序的分支结构



### ☼ 单分支结构微实例

#### ☼ PM2.5空气质量提醒①

PM2.5数值在0~35为优, 35~75为良, 75~115为轻度污染, 115~150为中度污染, 150~250为重度污染, 250~500为严重污染。

输入: PM2.5值

处理:

如果 $PM2.5 < 35$ ,打印空气质量优, 建议户外活动

如果 $35 \leq PM2.5 < 75$ ,打印空气质量良, 建议适度户外活动

如果 $PM2.5 \geq 75$ ,打印空气污染警告, 建议佩戴口罩

输出: 空气质量提醒

问题IPO描述

```
# 4.1PM2.5Warning
PM = eval(input("请输入PM2.5数值: "))
if 0 <= PM < 35:
    print("空气质量优, 建议户外活动。")
if 35 <= PM < 75:
    print("空气质量良, 适度户外活动。")
if 75 <= PM:
    print("空气污染, 出门时请佩戴口罩。")
```

Python代码描述





## 4.2 程序的分支结构



### 二分支结构：if-else语句

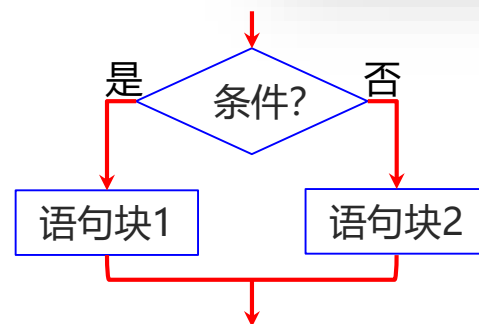
#### 语法格式

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

一般格式

```
<语句块1> if <条件> else <语句块2>
```

简洁格式



二分支结构

```
>>> count = 2  
>>> if count % 2 == 0:  
    print("偶数")  
else:  
    print("奇数")  
偶数  
>>> "偶数" if count % 2 == 0 else "奇数"  
'偶数'
```



## 4.2 程序的分支结构



### 二分支结构微实例

#### PM2.5空气质量提醒②

PM2.5数值在0~75为没有污染，75~500为存在污染。

```
# 4.2PM2.5Warning
PM = eval(input("请输入PM2.5数值: "))
if PM >= 75:
    print("空气存在污染, 出门时请佩戴口罩。")
else:
    print("空气没有污染, 可以户外活动。")

print("空气{}污染! ".format("存在" if PM >= 75 else "没有"))
```

Python代码描述



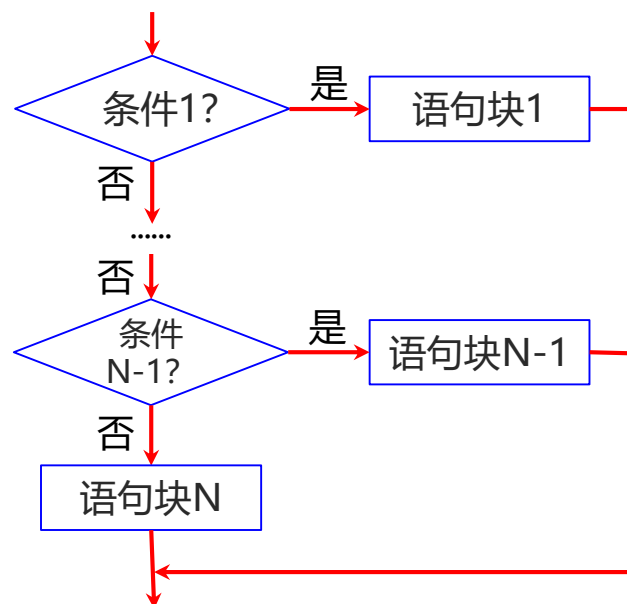
## 4.2 程序的分支结构



### 多分支结构：if-elif-else语句

#### 语法格式

```
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



## 4.2 程序的分支结构



### 多分支结构微实例

#### PM2.5空气质量提醒③

PM2.5数值在0~35为优，35~75为良，75以上为存在污染。

```
# 4.3PM2.5Warning
PM = eval(input("请输入PM2.5数值: "))
if 0 <= PM < 35:
    print("空气质量优, 建议户外活动。")
elif 35 <= PM < 75:
    print("空气质量良, 适度户外活动。")
else:
    print("空气污染, 出门时请佩戴口罩。")
```

Python代码描述

```
# 4.1PM2.5Warning
PM = eval(input("请输入PM2.5数值: "))
if 0 <= PM < 35:
    print("空气质量优, 建议户外活动。")
if 35 <= PM < 75:
    print("空气质量良, 适度户外活动。")
if 75 <= PM:
    print("空气污染, 出门时请佩戴口罩。")
```

单分支结构代码



## 4.3 实例：身体质量指数BMI



## 4.3 实例：身体质量指数BMI



### 身体质量指数

BMI = 体重(kg) / 身高<sup>2</sup>(m<sup>2</sup>)

客观衡量人的肥胖程度或健康程度。

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



## 4.3 实例：身体质量指数BMI



### 身体质量指数

- 编写根据体重和身高计算BMI值的程序，同时输出国际和国内的BMI指标建议值。

输入：身高和体重值

处理：计算BMI值，并根据BMI指标分类找到合适类别

输出：打印指标分类信息

问题IPO描述

```
# 4.4CalculateBMI.v1.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, dom = "", ""
if bmi < 18.5: # WHO标准
    who = "偏瘦"
elif bmi < 25: # 18.5 ≤ bmi < 25
    who = "正常"
elif bmi < 30: # 24 ≤ bmi < 30
    who = "偏胖"
else:
    who = "肥胖"
```

```
if bmi < 18.5: # 卫生部标准
    dom = "偏瘦"
elif bmi < 24: # 18.5 ≤ bmi < 24
    dom = "正常"
elif bmi < 28: # 24 ≤ bmi < 28
    dom = "偏胖"
else:
    dom = "肥胖"

print("BMI指标为:国际{}, 国内{}".format(who,
dom))
```

Python代码描述



## 4.3 实例：身体质量指数BMI



### 身体质量指数

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

```
# 4.4CalculateBMI.v2.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, dom = "", ""
if bmi < 18.5:
    who, dom = "偏瘦", "偏瘦"
elif 18.5 <= bmi < 24:
    who, dom = "正常", "正常"
elif 24 <= bmi < 25:
    who, dom = "正常", "偏胖"
elif 25 <= bmi < 28:
    who, dom = "偏胖", "偏胖"
elif 28 <= bmi < 30:
    who, dom = "偏胖", "肥胖"
else:
    who, dom = "肥胖", "肥胖"
print("BMI指标为:国际{}, 国内{}".format(who, dom))
```

Python代码描述





## 4.4 程序的循环结构

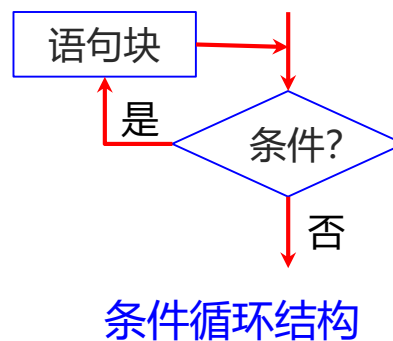
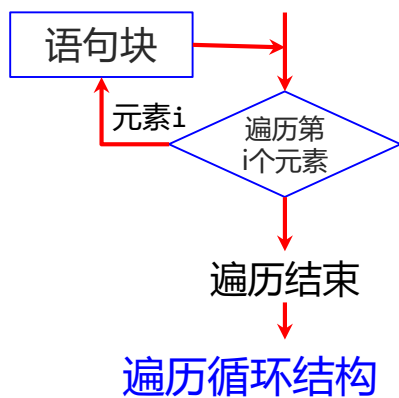


## 4.4 程序的循环结构



### 🔗 循环结构

- 🔗 确定次数循环→遍历循环 (for)
- 🔗 非确定次数循环→条件循环 (while)





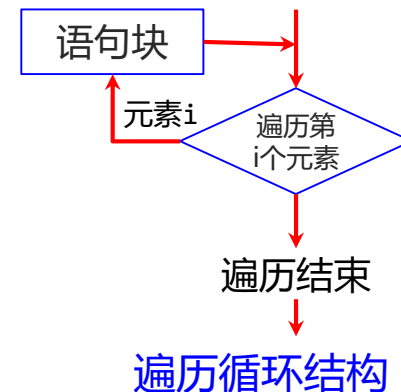
## 4.4 程序的循环结构



### 遍历循环：for语句

#### 语法格式

```
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并放在循环变量中，对于所提取的每个元素执行一次语句块
- 遍历结构可以是字符串、文件、组合数据类型（列表、字典等）、range()函数等 可迭代对象



## 4.4 程序的循环结构



### 遍历循环

遍历字符串s

```
for c in s:  
    <语句块>
```

遍历文件fi

```
for line in fi:  
    <语句块>
```

遍历列表ls

```
for item in ls:  
    <语句块>
```

循环N次

```
for i in range(N):  
    <语句块>
```

### 扩展模式

```
for <循环变量> in <遍历结构>:  
    <语句块1>  
else:  
    <语句块2>
```

for循环正常执行后继续执行else语句

```
for s in "BIT":  
    print("循环进行中：" + s)  
else:  
    s = "循环正常结束"  
print(s)
```

运行结果：

循环进行中：B  
循环进行中：I  
循环进行中：T  
循环正常结束

for循环示例



## 4.4 程序的循环结构



### 无限循环：while语句

- 根据条件进行循环，又称条件循环
- 无限循环一直保持循环操作直到不满足循环条件，无需提前确定循环次数。

### 语法格式

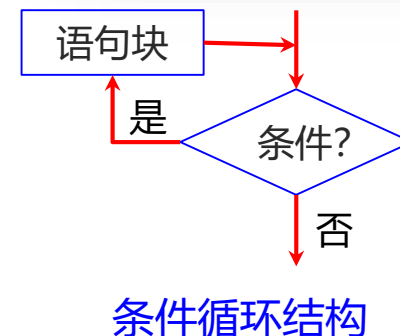
```
while <条件>:  
    <语句块>
```

扩展模式

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

```
s, idx = "BIT", 0  
while idx < len(s):  
    print("循环进行中：" + s[idx])  
    idx += 1  
else:  
    s = "循环正常结束"  
print(s)  
运行结果：  
循环进行中：B  
循环进行中：I  
循环进行中：T  
循环正常结束
```

while循环示例





## 4.4 程序的循环结构



### 🔗 循环保留字 辅助控制循环执行

#### 🔗 break

🔗 跳出最内层for或while循环（终止整个当前循环）

#### 🔗 continue

🔗 结束当前当次循环，不执行其后语句，直接进入下次循环

```
for s in "BIT":  
    for i in range(10):  
        print(s, end='')  
        if s == 'I':  
            break
```

运行结果：

BBBBBBBBBBITTTTTTTTTT

break跳出当前循环示例

```
for s in "BIT":  
    for i in range(10):  
        if s == 'I':  
            continue  
        print(s, end='')
```

运行结果：

BBBBBBBBBBTTTTTTTTTT

continue跳过本次循环示例

```
if s == 'I':  
    pass  
else:  
    print(s, end='')
```



## 4.4 程序的循环结构



### 🔗 循环保留字与循环else扩展用法

- 🔗 continue保留字对else的执行没有影响
- 🔗 break和return保留字使else语句不执行

else执行条件:

正常遍历所有内容或循环条件不成立而结束循环。

```
for s in "PYTHON":  
    if s == "T":  
        continue  
    print(s, end='')  
else:  
    print("正常退出")
```

运行结果:

PYHON正常退出

continue不影响else

```
for s in "PYTHON":  
    if s == "T":  
        break  
    print(s, end='')  
else:  
    print("正常退出")
```

运行结果:

PY

break跳过else



## 4.5 random库的使用









## 4.5 random库的使用



### random库

-  Python解释器内置的标准库
-  采用梅森旋转算法生成伪随机数
-  可用于除随机性要求更高的加解密算法外的大多应用
-  基本用法

```
import random  
random.random()
```



## 4.5 random库的使用



### 9个随机数生成函数

函 数	描 述
<code>seed(a=None)</code>	初始化随机种子，默认为当前系统时间
<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]之间的随机整数
<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>	从序列类型中随机返回一个元素
<code>shuffle(seq)</code>	将序列类型中的元素随机排列，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，以列表形式返回



## 4.5 random库的使用



### 使用示例

```
>>> from random import *
>>> random()
0.05459719768975391
>>> randint(1,10)
8
>>> randrange(0,100,4)
84 #从0开始以4递增的元素
>>> uniform(1,10)
2.3854671931138913
>>> choice("python")
'o'
>>> choice(range(100))
70
>>> ls = list(range(10))
>>> shuffle(ls)
>>> ls
[5, 6, 4, 1, 8, 9, 2, 7, 0, 3]
>>> sample("python", 3)
['t', 'n', 'o']
```

相同随机数种子生成的随机数是相同的。

```
>>> seed(125)
>>>
"{}.{}.{}".format(randint(1,10),
randint(1,10),randint(1,10))
'4.4.10'
>>>
"{}.{}.{}".format(randint(1,10),
randint(1,10),randint(1,10))
'5.10.3'
>>> seed(125)
>>>
"{}.{}.{}".format(randint(1,10),
randint(1,10),randint(1,10))
'4.4.10'
>>>
"{}.{}.{}".format(randint(1,10),
randint(1,10),randint(1,10))
'5.10.3'
```







## 4.6 $\pi$ 值的计算

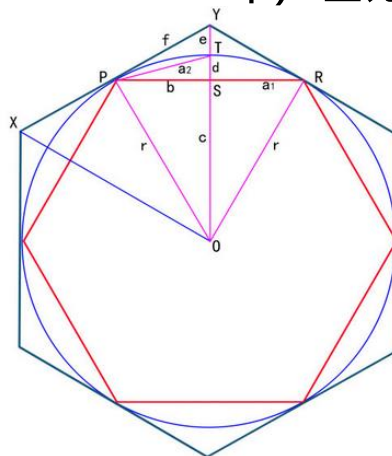


## 4.6 $\pi$ 值的计算



### $\pi$ 值计算

-  古巴比伦石匾（约BC1900年）：圆周率 =  $25/8 = 3.125$
-  古埃及莱因德数学纸草书：圆周率 =  $16/9$  的平方  $\approx 3.1605$
-  古印度《百道梵书》（约BC800年）：圆周率 =  $339/108 \approx 3.139$
-  古希腊阿基米德(BC287–212 年) 理论计算圆周率：3.141851







阿基米德从单位圆出发，先用内接正多边形求出圆周率的下界，再用外接正多边形求出圆周率的上界。

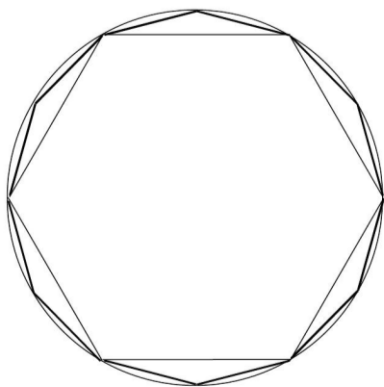


## 4.6 $\pi$ 值的计算



### $\pi$ 值计算

-  中国《周髀算经》（约BC200）：圆周率=3
-  汉朝张衡：圆周率= $\sqrt{10} \approx 3.162$
-  魏晋刘徽(公元263年)“割圆术”：圆周率= $3927/1250 \approx 3.1416$
-  南北朝祖冲之（公元480年）：圆周率在3.1415926和3.1415927之间；两个近似分数值，密率355/113和约率22/7



割之弥细  
失之弥少  
割之又割  
以至于不可割  
则与圆合体  
而无所失矣

14

“割圆术”



祖冲之



## 4.6 $\pi$ 值的计算

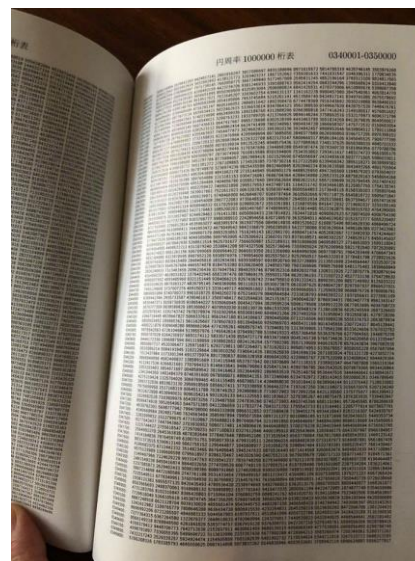
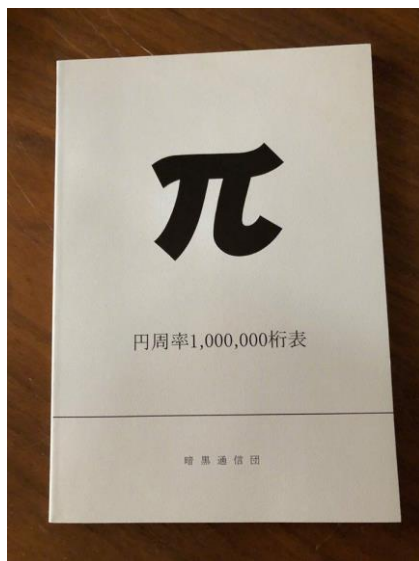


### $\pi$ 值计算

 现代方法：贝利-波尔温-普劳夫公式(BBP公式)

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

计算圆周率 $\pi$ 的第 $n$ 位二进制数







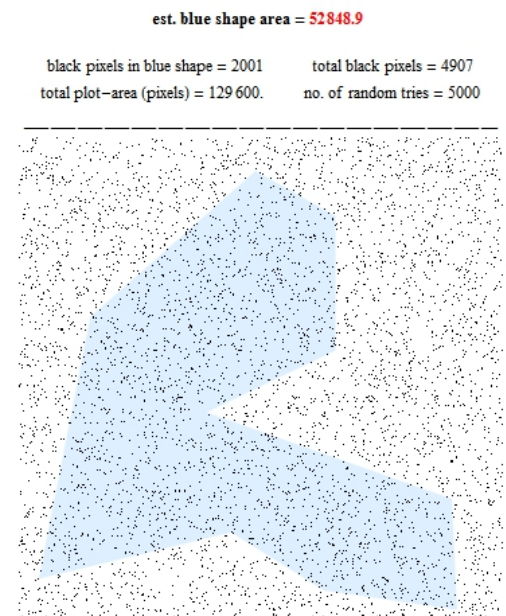
## 4.6 $\pi$ 值的计算



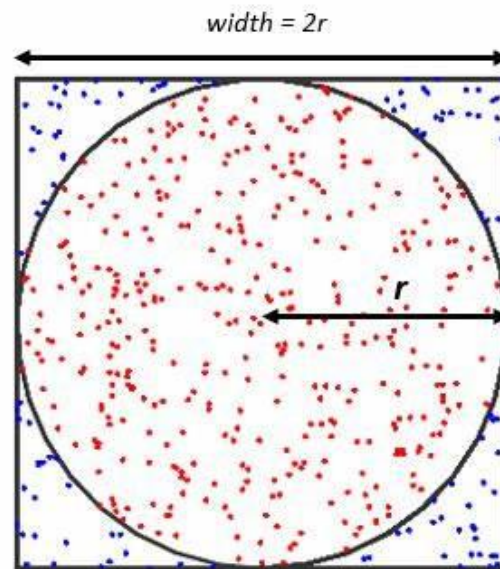
### π值计算

#### 概率统计方法

#### 蒙特卡罗 (Monte Carlo) 方法



蒙特卡罗方法计算面积



蒙特卡罗方法计算 $\pi$

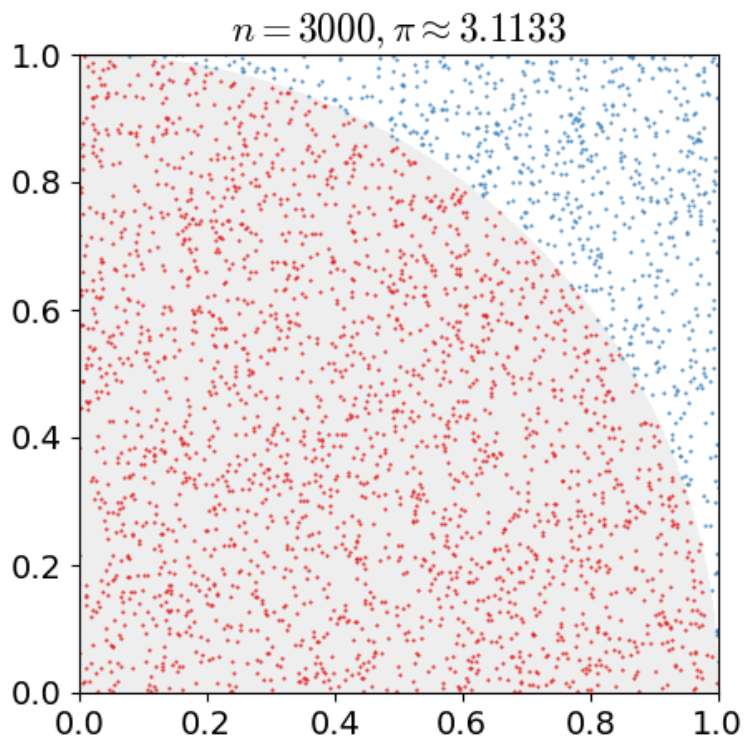




## 4.6 $\pi$ 值的计算



### 蒙特卡罗方法计算 $\pi$



蒙特卡罗方法计算 $\pi$   
使用的1/4区域和抛点过程

蒙特卡罗方法计算 $\pi$ 的基本步骤：

- 随机向边长为1的单位正方形内抛洒飞镖点；
- 计算飞镖点到圆心距离，距离 $\leq 1$ 在圆内，距离 $> 1$ 在圆外；
- 圆内飞镖点数除以总飞镖点数为 $\pi/4$ ；
- 随机飞镖点数量越大，所得 $\pi$ 值越精确。



## 4.6 $\pi$ 值的计算



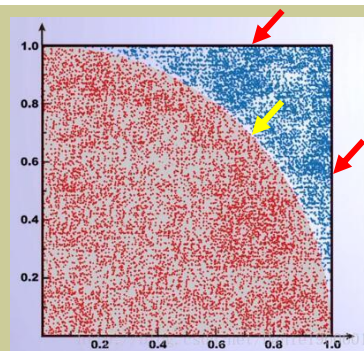
### 蒙特卡罗方法计算 $\pi$

输入：抛点数

处理：计算每个点到圆心距离，统计在圆内点的数量

输出： $\pi$ 值

```
# 4.5CalculatePi.py
from random import random
from math import sqrt
from time import time
DARTS = 100000000
hits = 0
start = time()
for i in range(1, DARTS+1):
    x, y = random(), random()
    dist = sqrt(x**2 + y**2)
    if dist <= 1.0:
        hits += 1
pi = 4 * (hits / DARTS)
print("Pi值是{}".format(pi))
print("运行时间是：{:0.5f}s".format(time()-start))
```



问题IPO描述

Python代码

(运行结果)  
Pi值是3.14170128.  
运行时间是： 58.28203s



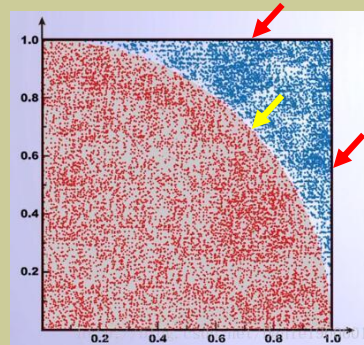
## 4.6 $\pi$ 值的计算



### 蒙特卡罗方法计算 $\pi$

取值边界问题

```
# 4.5CalculatePi.py
from random import uniform
from time import time
DARTS = 100000000
hits = 0
start = time()
for i in range(1, DARTS+1):
    x, y = uniform(0.0, 1.0), uniform(0.0, 1.0) ←
    if x**2 + y**2 <= 1.0:
        hits += 1
pi = 4 * (hits / DARTS)
print("Pi值是{}".format(pi))
print("运行时间是: {:.5f}s".format( time()-start))
```



Python代码

(运行结果)

Pi值是3.14148744.  
运行时间是: 78.85700s



## 4.6 $\pi$ 值的计算



### 蒙特卡罗方法计算 $\pi$

表 4.4 不同抛点数产生的精度和运行时间

DARTS	$\pi$	运行时间
$2^{10}$	3.109 375	0.011 s
$2^{11}$	3.138 671	0.012 s
$2^{12}$	3.150 390	0.014 s
$2^{13}$	3.143 554	0.018 s
$2^{14}$	3.141 357	0.030 s
$2^{15}$	3.147 827	0.049 s
$2^{16}$	3.141 967	0.116 s
$2^{18}$	3.144 577	0.363 s
$2^{20}$	3.142 669 677 7	1.255 s
$2^{25}$	3.141 697 883 6	40.13 s



## 4.7 程序异常处理



## 4.7 程序异常处理



### 异常处理

```
>>> num = eval(input("请输入一个整数: "))
请输入一个整数: 100
>>> print(num**2)
10000
>>> num = eval(input("请输入一个整数: "))
请输入一个整数: NO
Traceback (most recent call last):
  File "<pyshell#47>", 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
>>> int("num")
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    int("num")
ValueError: invalid literal for int() with base 10: 'num'
```



## 4.7 程序异常处理



### 异常处理

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

```
请输入一个整数: 100
```

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

```
10000
```

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

```
请输入一个整数: NO
```

```
Traceback (most recent call last):
```

```
File "<pyshell#47>", 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
```

```
File "test.py", line 2
```

```
if True
```

```
^
```

```
SyntaxError: invalid syntax
```

异常文件路径

异常发生的  
代码行数

异常内容提示

异常回溯  
标记

异常类型



## 4.7 程序异常处理



### 异常处理：try-except语句

#### 语法格式

```
try:
    <语句块1>
except <异常类型>:
    <语句块2>
```

当语句块1发生异常时执行except保留字后的语句块2。如果语句块1正常执行则不执行语句块2。

#### 增加异常处理

```
try:
    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! ")
```

(运行结果)  
请输入一个整数: NO  
输入错误, 请输入一个整数!





## 4.7 程序异常处理



### 异常处理：try-except语句

#### 高级用法

```
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>
```

类似于if-elif-else语句。  
相应except语句中的语句块  
只处理包含相应类型的异常。

最后一个except语句没有指  
定任何类型，对应语句块处  
理所有其他异常。

#### 完善异常处理

```
try:
    alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    idx = eval(input("请输入一个整数[1-26]: "))
    print(alp[idx-1])
except NameError:
    print("输入错误，请输入一个整数！")
except:
    print("其他错误")
```

(运行结果)

请输入一个整数[1-26]: NO  
输入错误，请输入一个整数！  
请输入一个整数[1-26]: 100  
其他错误



## 4.7 程序异常处理



### 异常处理：try-except语句

#### 扩展模式

```
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>
```

语句块1正常执行：  
try-else-finally  
语句块1发生异常：  
try-except-finally



图 4.14 异常处理控制流过程



## 4.7 程序异常处理



### 异常处理：try-except语句

#### 扩展模式

```
try:
    alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    idx = eval(input("请输入一个整数[1-26]: "))
    print(alp[idx+1])
except NameError:
    print("输入错误, 请输入一个整数! ")
else:
    print("没有发生异常")
finally:
    print("程序执行完毕, 不知道是否发生了异常")
```

(运行结果)





请输入一个整数[1-26]: 5  
E  
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[1-26]: 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



## 4.7 程序异常处理



### 异常处理：try-except语句

-  Python能识别多种异常类型，但不建议过度依赖try-except处理机制；
-  一般只用来检测极少发生的情况
  -  用户输入的合规性、文件是否成功打开
-  尽量使用if语句做判断



# 本章要点



## Python程序控制结构

### 程序的分支结构

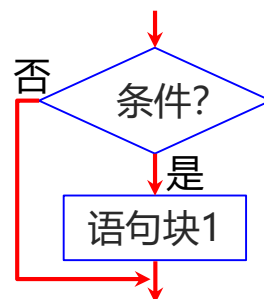
if-[elif-]else

### 程序的循环结构

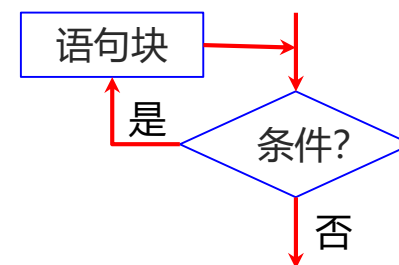
for、while

### random获得随机数

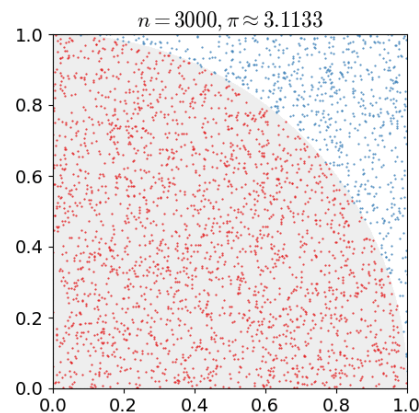
### 蒙特卡罗方法计算 $\pi$ 值



单分支结构



条件循环结构





# 程序练习题



## 4.1 猜数游戏

在程序中预设一个1-100的随机整数，让用户通过键盘输入所猜数，若大于预设数，显示“遗憾，太大了”；若小于预设数，显示“遗憾，太小了”，如此循环，直至猜中该数，显示“预测N次，恭喜你猜中了！”。要求当用户输入非整数时，给出“输入内容必须为整数！”的提示，并让用户重新输入。 代码文件名：4.1GuessNumber.py

## 4.2 统计不同字符个数

用户从键盘输入一行字符，编写一个程序，统计并输出英文字符、数字、空格和其他字符的个数。 代码文件名：4.2ChrCount.py



# 程序练习题



## 4.3 最大公约数和最小公倍数计算

从键盘接受2个整数，编写程序求出这两个整数的最大公约数和最小公倍数（提示：最小公倍数等于两数之积除以其最大公约数）。

代码文件名：4.3CalGCDLCM.py

.py代码文件打包(4.学号+姓名)发送到  
python\_xxmu@163.com



# 下周课程

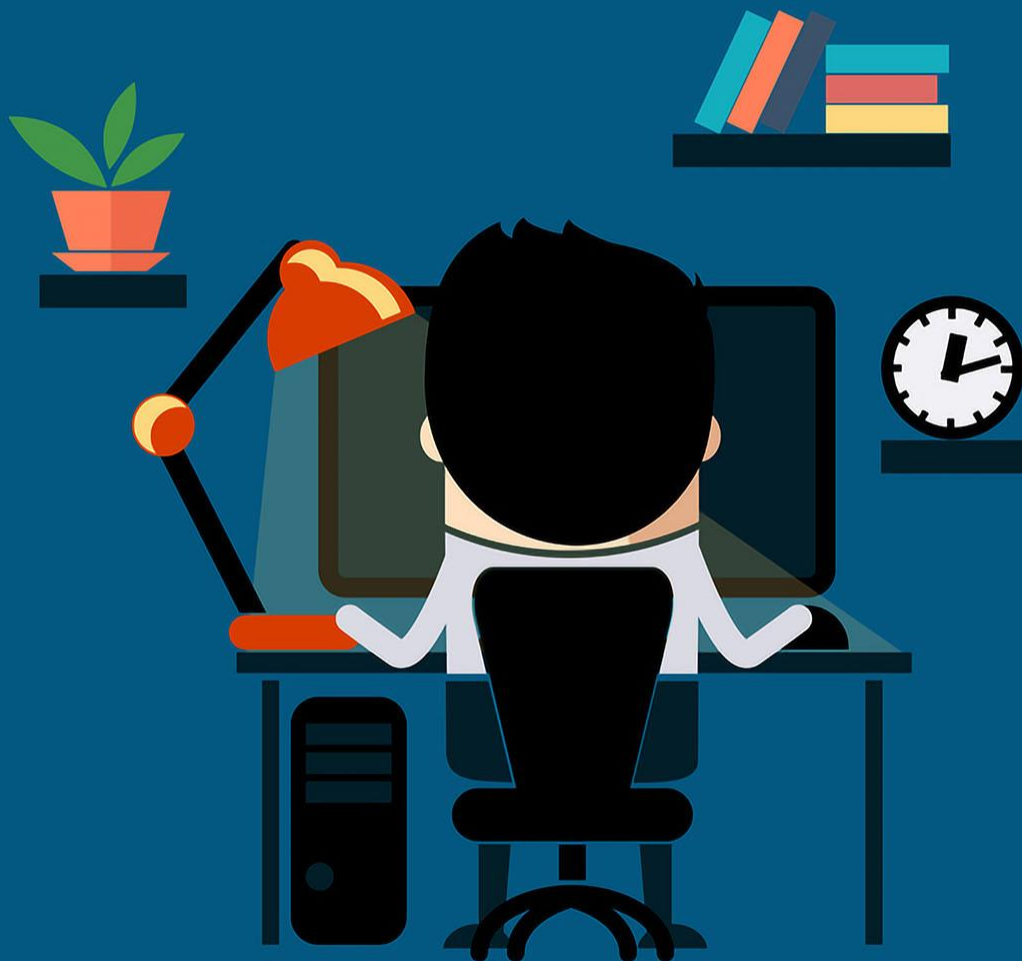


## 🔗 第5章 Python函数与模块

- 🔗 函数的基本使用
- 🔗 函数的参数传递
- 🔗 datetime库的使用
- 🔗 实例：七段数码管绘制
- 🔗 代码复用和模块化设计
- 🔗 函数的递归
- 🔗 实例：科赫曲线绘制



编程辣么好，还等什么？开始学习吧！



Programing is an Art