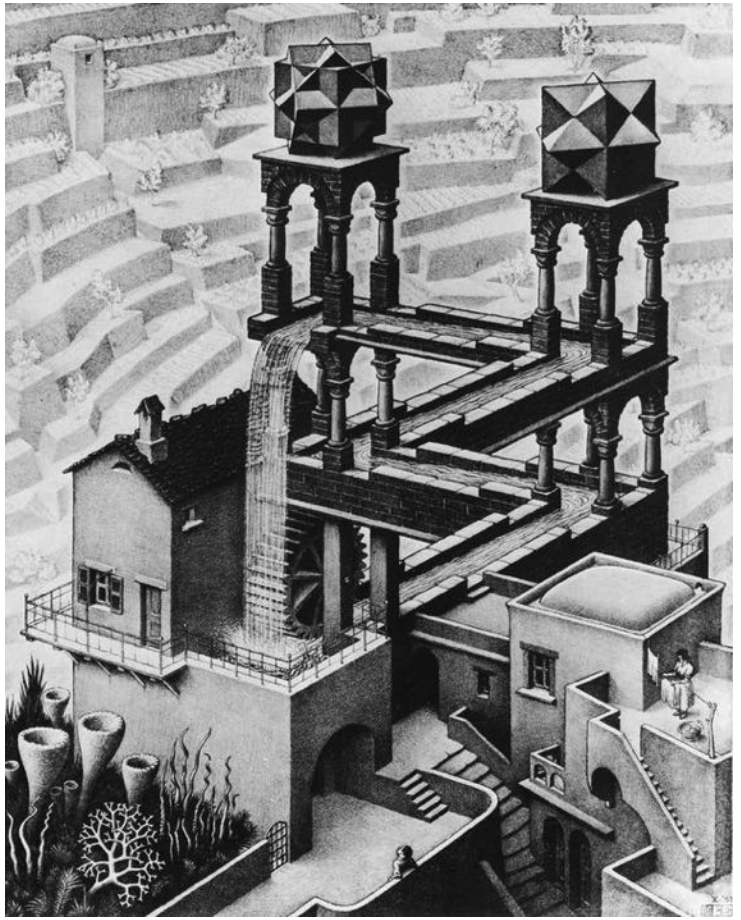
Python语言基础与应用03

北京大学 陈斌

2019.07.03

目录

- 重复：循环结构
- 迭代循环：for语句
  - 常用的迭代数据集
- 绘制数学曲线
- 条件分支：if语句
- 条件循环：while语句
- break和continue
- 迭代与枚举
- 算法分析



Python语言的几个要件

数据对象和组织

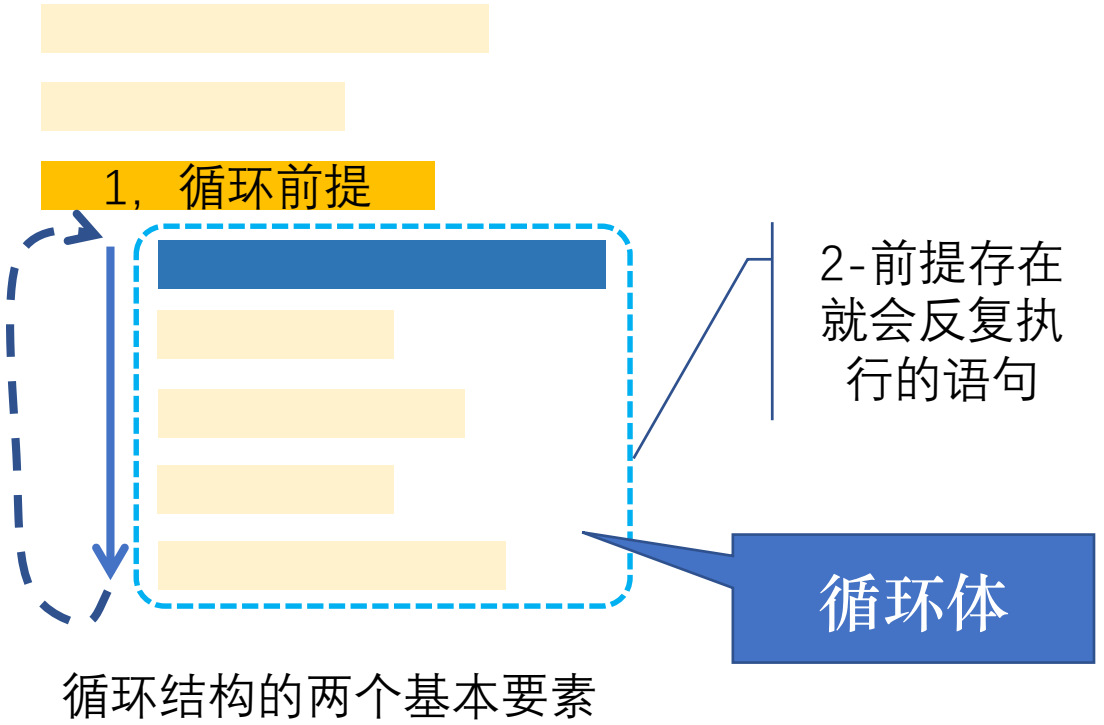
- 对现实世界实体和概念的抽象
- 分为简单类型和容器类型
- 简单类型用来表示值
  - 整数int、浮点数float、复数complex、逻辑值bool、字符串str
- 容器类型用来组织这些值
  - 列表list、元组tuple、集合set、字典dict
- 数据类型之间几乎都可以转换

赋值和控制流

- 对现实世界处理和过程的抽象
- 分为运算语句和控制流语句
- 运算语句用来实现处理与暂存
  - 表达式计算、函数调用、赋值
- 控制流语句用来组织语句描述过程
  - 顺序、条件分支、循环
- 定义语句也用来组织语句，描述一个包含一系列处理过程的计算单元
  - 函数定义、类定义

重复：循环结构（loop）

- 我们需要让计算机反复做设定的任务
- 又能在该停止的时候自动停止重复
- 循环结构具有两个要素
  - 一个循环前提
  - 一组重复执行的语句（循环体）
- 只要循环前提成立，循环体就会被反复执行



循环结构的两个基本要素

迭代循环：for语句

- 迭代循环语句：for语句
- 循环前提：
  - 一个（或一组）循环变量
  - 一个数据对象集
- for语句每次从对象集中取出一个数据对象，赋值给循环变量
  - 如果能取到，就执行一次循环体
    - 循环体中可以使用循环变量
  - 如果取完了，就退出循环

forst.py - /Users/chenbin/Documents/教学项目/北大附

```
s = 0
for i in range(1, 101):
    s = s + i
print("sum:1..100:", s)

a, b, c = 10, -5, 0.5
for x in [-25, 0, 10, 35, 100]:
    y = a * x * x + b * x + c
    print (f"f({x})={y}")

for name in ["Tom", "Jerry", "张三"]:
    print("Hello!", name)
```

常用的数据集：range函数

- range函数可以产生连续整数构成的数据集
- range(end)
  - [0, end)
- range(start, end)
  - [start, end)
- range(start, end, step)
  - [start, end) 步长step
  - 如果step小于0则反向取

range()函数产生一个连续整数的数据集

range(end)

range(start, end)

range(start, end, step)

```
1   print("range:5")
2   for i in range(5):
3       print(i)
4
5   print("range:1,5,2")
6   for i in range(1, 5, 2):
7       print(i)
8
9   print("range:4,1,-1")
10  for n in range(4, 1, -1):
11      print(n)
```

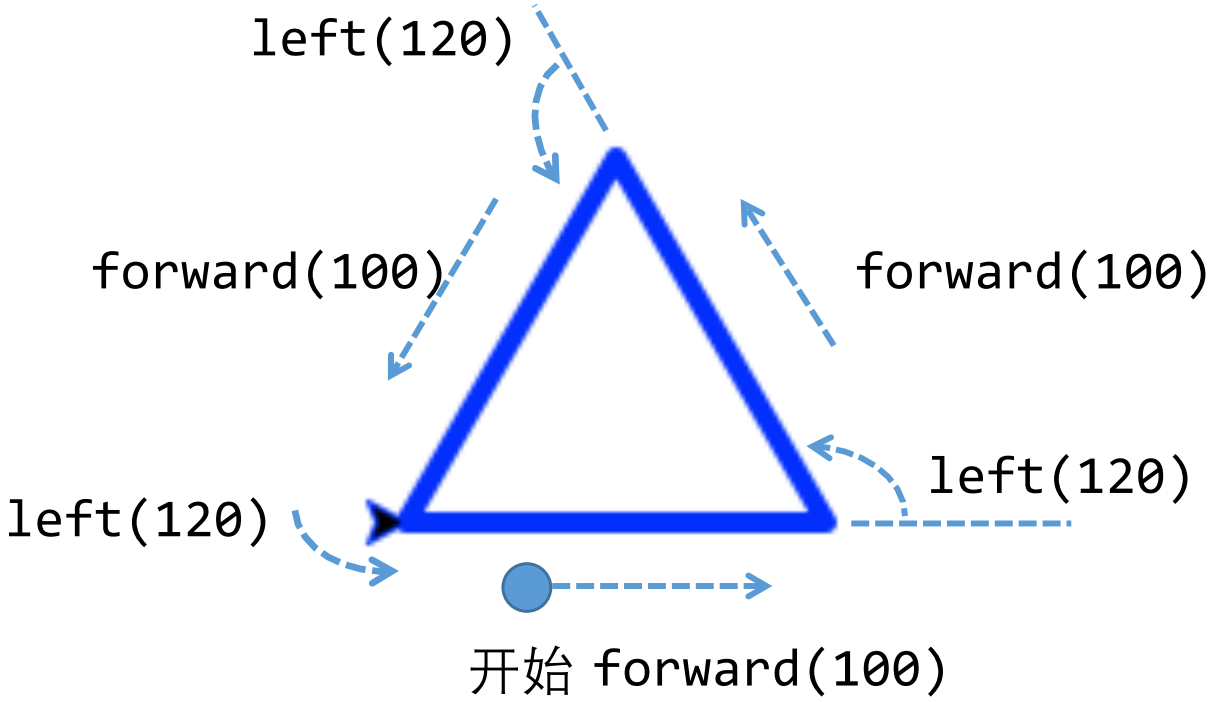
常用的数据集：列表list

- 列表是一种容器数据类型，可以包容多个数据对象
- 整数/浮点数列表
  - `[1, 3, 5, 35, -10]`
  - `[1.23, 34.5, 10.0, 245.7]`
- 字符串列表
  - `["Tim", "Jay", "Mary"]`
- 混合列表
  - `["Hello", True, 12, 34.56]`

```
for name in ["Tom", "Jerry", "张三"]:
    print("Hello!", name)

for n in [1, 3, 5, 35, -10]:
    print(f"{n}^2=", n * n)

m = 1
for f in [1.23, 34.5, 10.0, 245.7]:
    m *= f
print("product:", m)

for k in [12, 30, 8, 10, 9]:
    print(f"{k:02d}>", "#"*k)
```

练习：循环语句的应用

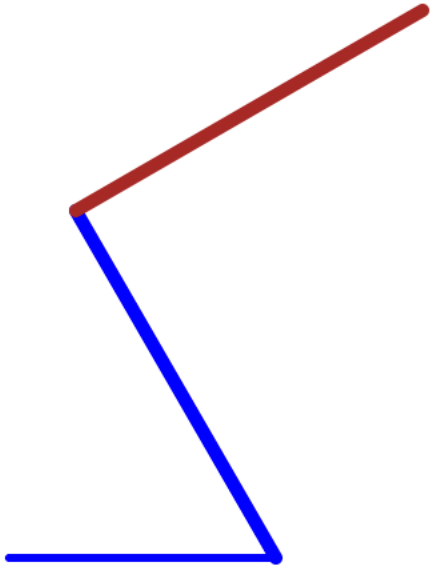
- 如何用循环语句画出等边三角形?
- 如何用循环语句画出一个4种颜色边的正方形?



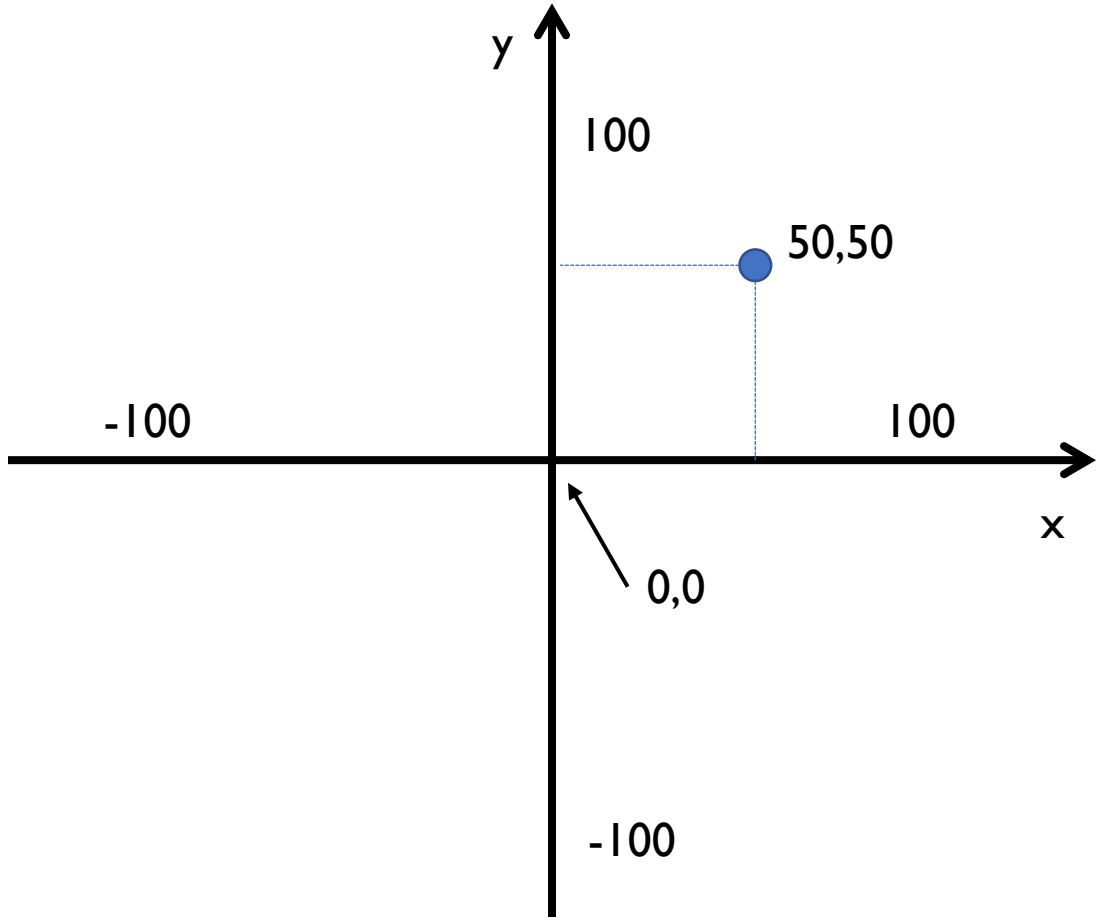
作图程序模版

- 首先，导入turtle模块
- 然后，生成一只海龟
  - 可以做一些初始化设定
- 程序主体：用作图语句绘图
- 最后结束作图
  - 可选隐藏海龟：t.hideturtle()

```
1   # 1. 导入海龟模块
2   import turtle
3
4   # 2. 生成一只海龟，做一些设定
5   t = turtle.Turtle()
6   t.color("blue")
7   t.pensize(3)
8
9   # 3. 用海龟作图
10  t.forward(100)
11  t.right(60)
12  t.pensize(5)
13  t.backward(150)
14  t.left(90)
15  t.color("brown")
16  t.forward(150)
17
18  # 4. 结束作图
19  t.hideturtle()
20  turtle.done()
```



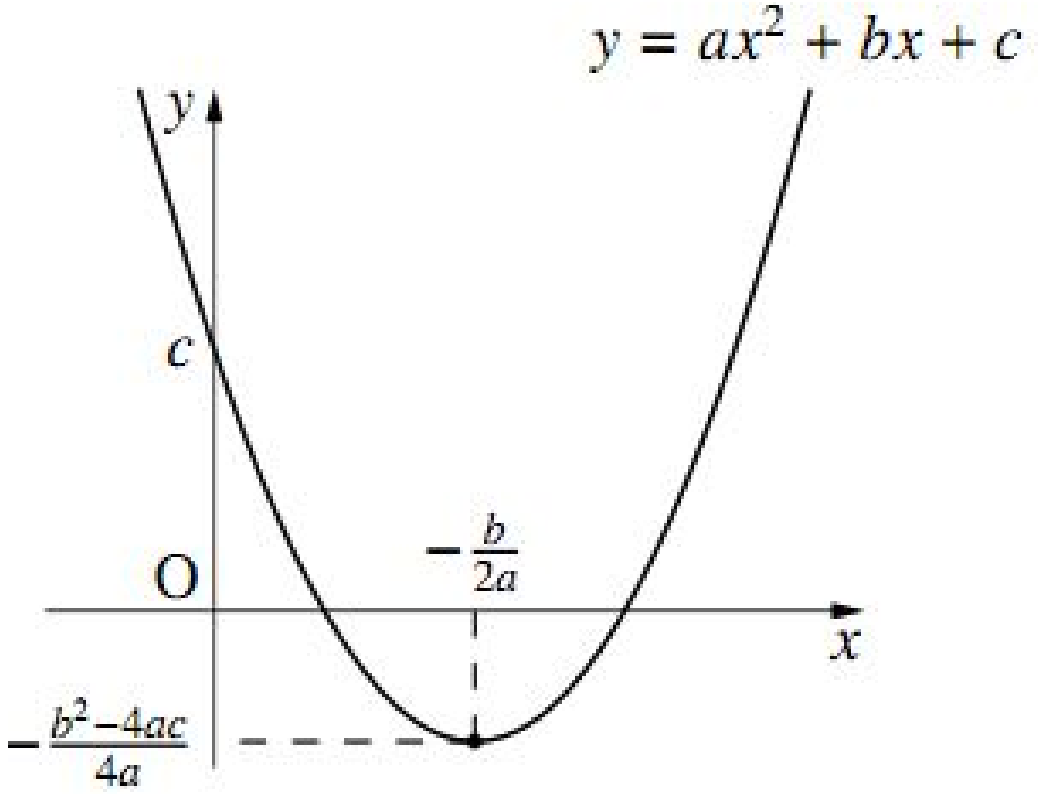
让海龟直接去到某个位置goto(x,y)



- 最开始海龟在 (0,0)
- 可以用goto让海龟去到任何地方
- 用position()获取海龟当前位置
- 如果不希望留下轨迹，则需要在goto之前调用penup()

平面直角坐标系曲线绘制

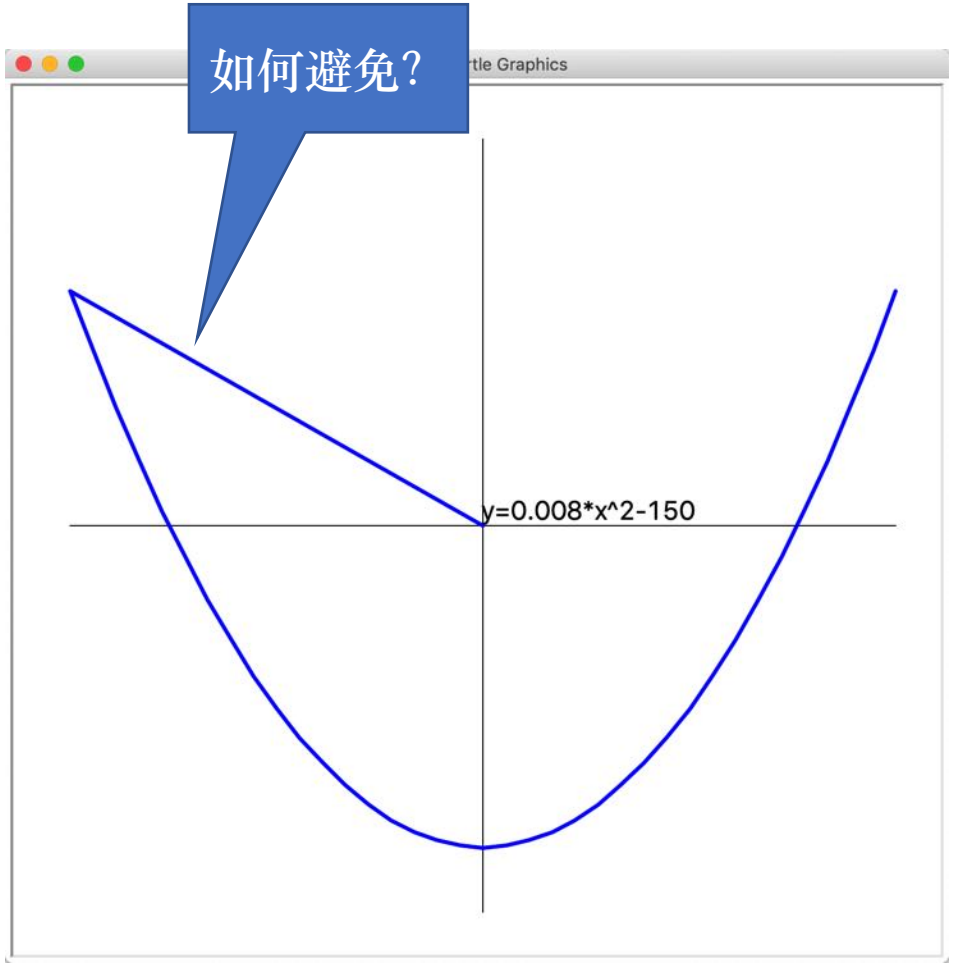
- 如何用循环语句绘制数学曲线？
  - y=ax+b、$y=ax^2+bx+c$、y=sin(x)
- 一般步骤
  - 估计x,y的范围
  - 设定坐标系：左下角/右上角坐标
  - 画出坐标轴（可选：标注公式）
  - 迭代循环x，计算y
  - goto(x,y)将点连接起来
  - 可以叠加多条曲线



示例：绘制数学曲线

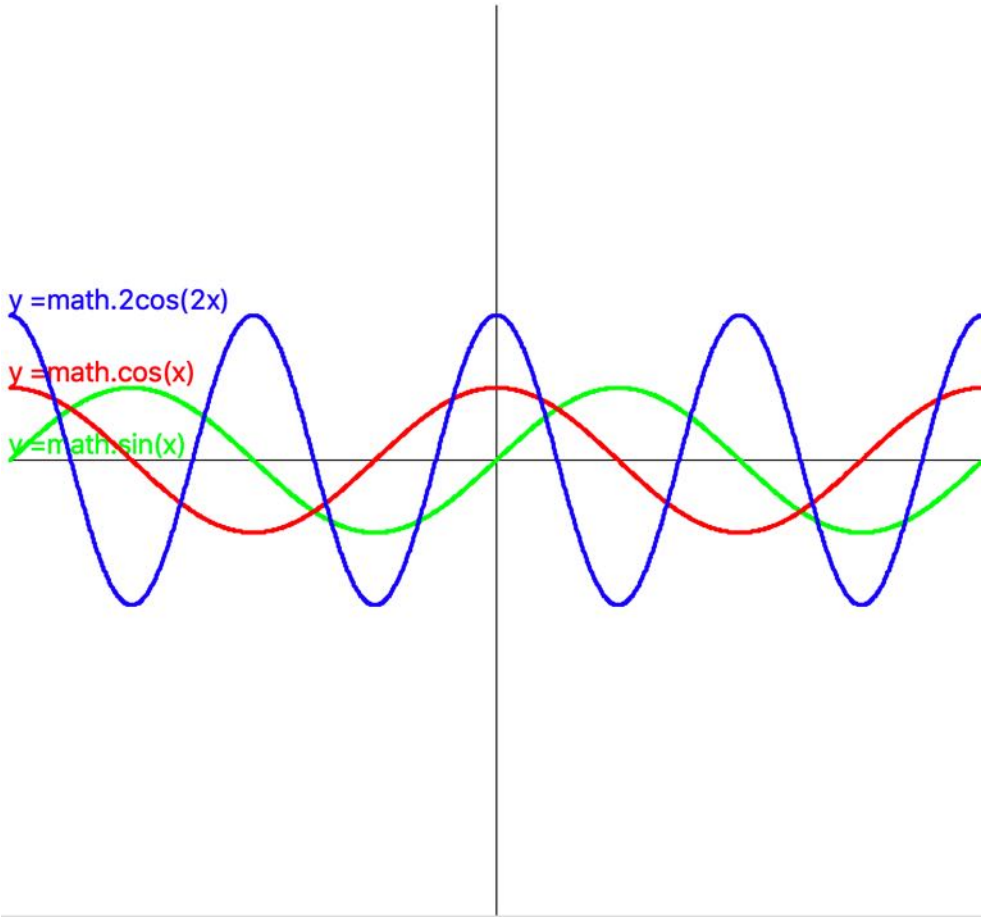
```
# 1. 导入海龟模块
import turtle

# 2. 生成一只海龟，做一些设定
t = turtle.Turtle()
# 设定坐标系：左下角x,y；右上角x,y
turtle.setworldcoordinates(-200, -200, 200, 200)

# 3. 用海龟作图
t.penup()
t.goto(-180, 0)
t.pendown()
t.goto(180, 0)
t.penup()
t.goto(0, -180)
t.pendown()
t.goto(0, 180)
t.goto(0, 0)
t.write("y=0.008*x^2-150", font=("consolas", 20, "normal"))
t.pencolor("blue")
t.pensize(3)
for x in range(-180, 181, 10):
    y = 0.008 * x * x - 150
    t.goto(x, y)

# 4. 结束作图
t.hideturtle()
turtle.done()
```

如果x是小数怎么办?



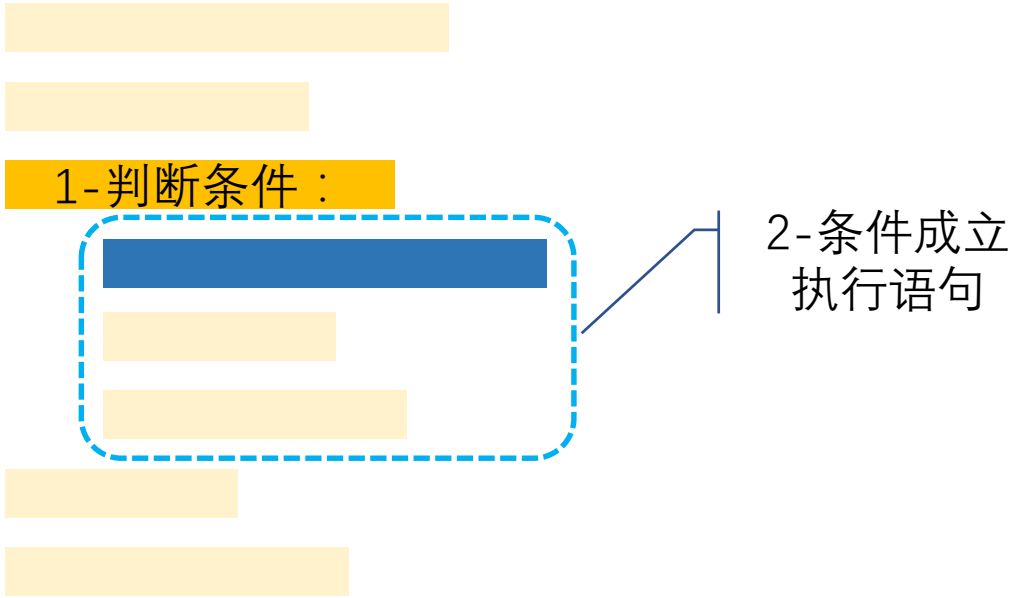
随堂作业：绘制三角函数曲线

- 写一个程序tri.py
- 提交希悦
- 叠加绘制下面3个函数
  - 绿色：y=sin(x)
  - 红色：y=cos(x)
  - 蓝色：y=2cos(2x)
  - x的范围是-2pi～2pi



条件分支结构：if语句

- 让计算机能够自动根据当前的状况来决定执行哪些语句
- 条件分支结构的2个要素
  - 判断条件
  - 一组语句
- if语句首先计算判断条件
  - 如果得到True，就执行这组语句
  - 否则，不执行



条件分支结构两个基本要素

```
# 判断偶数
n = int(input("n="))
print("Your number is", n)
if n % 2 == 0:
    print("It's a even number!")
```

if语句的附加要素：elif和else

- if语句可以附加两个子句
- else子句可以指定在判断条件不成立的时候，要执行的一组语句
- elif子句可以在判断条件不成立的时候，再继续判断另一个条件，相当于else：if

```
# 计算x1和x2之间的距离
x1 = int(input("x1="))
x2 = int(input("x2="))
if x1 > x2:
    d = x1 - x2
else:
    d = x2 - x1
print("distance=", d)
```

```
# 判断年龄
age = int(input("age="))
print("年龄: ", age)
if 0 <= age <= 6:
    print("童年")
elif 7 <= age <= 17:
    print("少年")
elif 18 <= age <= 40:
    print("青年")
elif 41 <= age <= 65:
    print("中年")
else:
    print("老年")
```

随堂作业：判断三角形/找最大数

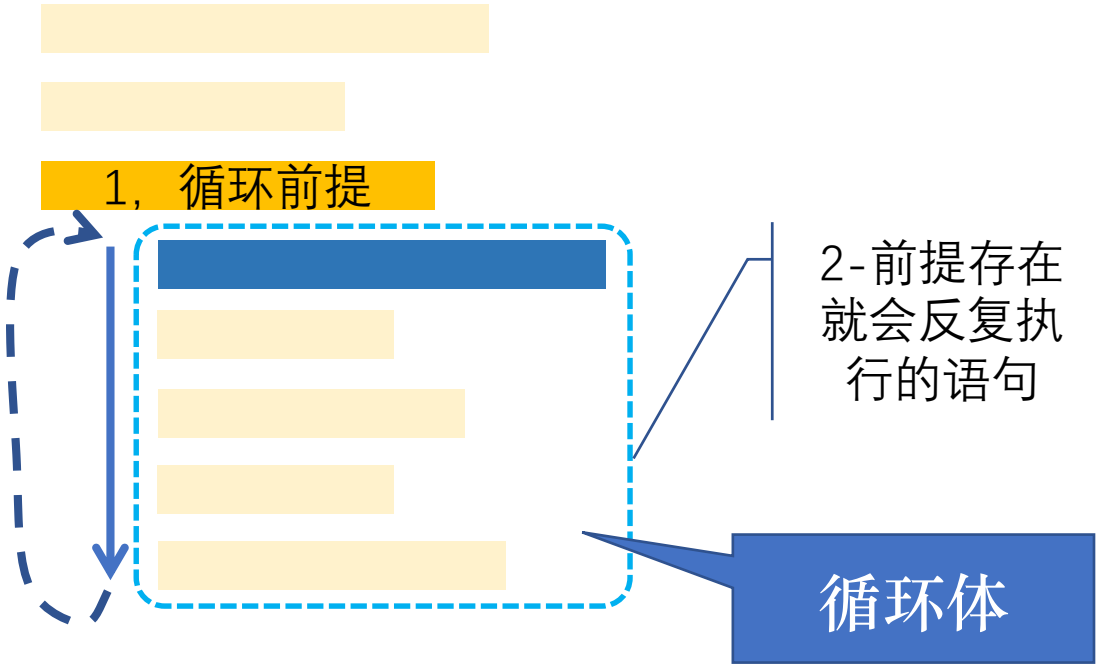
- 程序1（judge3.py)
  - 输入a,b,c三个整数
  - 输出这三个长度的边是否可以构成三角形
- 程序2（fmax.py)
  - 输入若干个数，用空格隔开
  - 输出这些数中的最大数

```
# 获取输入一行用空格隔开的整数
numbers = input("some numbers:").split()
numbers = list(map(int, numbers))
print(numbers)
```

```
# 简写
numbers = list(map(int, input().split()))
```

复习：循环结构（loop）

- 我们需要让计算机反复做设定的任务
- 又能在该停止的时候自动停止重复
- 循环结构具有两个要素
  - 一个循环前提
  - 一组重复执行的语句（循环体）
- 只要循环前提成立，循环体就会被反复执行



循环结构的两个基本要素

条件循环：while语句

- 循环前提是一个判断条件
  - 逻辑类型表达式
- while语句每次都计算表达式
  - 如果结果为“真”True，就执行循环体，然后再计算条件
  - 如果结果为“假”False，就退出循环
- 条件循环一般用在事先不确定循环的次数的情况
  - 但知道循环什么时候应该结束

```
# 当n等于多少的时候，n阶乘超过10亿?
n = np = 1
while np < 10**9:  # 不超过10亿就继续循环
    n = n + 1
    np = n * np
print(n, "阶乘超过10亿，等于", np)

# ========================================

# 第一个能同时整除2/3/4/5/6的整数是哪个?
n = 1
while not (n % 2 == 0 and n % 3 == 0 \
           and n % 4 == 0 and n % 5 == 0 \
           and n % 6 == 0):
    n = n + 1
print("第一个能同时整除2/3/4/5/6的整数是", n)
```

条件循环和条件分支有什么不同?

```
# 摇几次骰子能摇到6?
import random
n = 1
while random.randint(1, 6) != 6:
    n = n + 1
print("摇了", n, "次骰子得到6")
```

```
import random
n = 1
if random.randint(1, 6) != 6:
    n = n + 1
print("摇了", n, "次骰子得到6")
```

【练习】验证3x+1问题

- 有一个神奇的数学问题叫做“3x+1”问题
- 从任意一个正整数开始，重复对其进行下面的操作：
  - 如果这个数是偶数，把它除以2；
  - 如果这个数是奇数，则把它扩大到原来的3倍后再加1。
- 你会发现，序列最终总会变成4, 2, 1, 4, 2, 1, … 的循环

- 写一个程序x31.py
- 请用户输入一个正整数n
- 然后打印出整数变化序列
- 直到变成1为止
- 统计n变到1的变化次数step

【随堂作业】计算机模拟统计概率

- 写一个程序dice.py
- 请用户输入摇骰子次数n
- 让计算机模拟摇骰子n次
- 统计摇到6点的概率是多少?
- 试几个n，看看这个概率跟n有没有关系?

- 修改dice.py这个程序为dice2.py
- 不需要用户输入
- 让n从100开始，每次10倍，直到1亿
- 打印出n和对应的概率
- 看看变化规律?

2层嵌套循环

break语句

- 有时候需要立刻中断循环
- break语句立刻中断退出循环
  - 如果有多个循环嵌套，仅退出直接包含它的那一层循环
- 可以用在for和while循环语句中
- 我们试着用for + break语句重写阶乘的例子：

```
# 当n等于多少的时候，n阶乘超过10亿?
np = 1
for n in range(1, 100):
    np = np * n
    if np > 10**9:
        break
print(n, "阶乘超过10亿，等于", np)
```

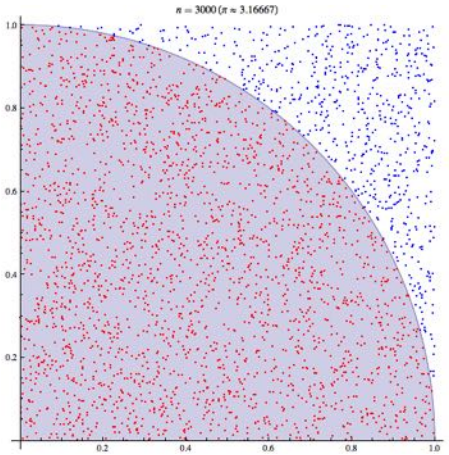
continue语句

- 有时候在执行循环体语句的时候，需要忽略余下的语句，直接跳到下一次循环
- continue语句立刻跳到下一次循环
  - 仅作用于直接包含它的循环语句
- 可以用在for和while语句
- 示例：过7
- 这个例子可以不用continue?

```
# ======================================

for i in range(1, 100):  # 从1到99的报数游戏
    # 包含数字7, 或者能被7整除
    if ('7' in str(i)) or (i % 7 == 0):
        print("过! ", end = ', ')
        continue
    print(i, end = ', ')
```

【H5】蒙特卡罗法求圆周率

- 蒙特卡罗是一类随机方法的统称。这类方法的特点是，可以在随机采样上计算得到近似结果，随着采样的增多，得到的结果是正确结果的概率逐渐加大



- 编写一个程序mcpi.py
  - random.uniform(0, 1)生成0～1之间的随机浮点数
  - 生成1万个落在正方形（0,0,1,1）里的点(x,y)
  - 用turtle画出正方形和1/4圆
    - 坐标系设置为(0,0,1,1)
    - turtle.setworldcoordinates(0, 0, 1, 1)
  - 把点标注在图形上
    - t.dot(1, 'red') # 在当前位置画点
    - 落在圆内用红色，圆外用蓝色
  - 计算出圆周率pi是多少?

问题求解策略：迭代与枚举

- 编程解决问题，最简单最基本的方法，是从所有可能的情况中找到答案，称为“枚举策略”
- 枚举策略解决问题的一般过程
  - ① 确定问题的解所包含的变量；
  - ② 确定每个变量可能的取值范围；
  - ③ 枚举所有变量的取值组合；
  - ④ 对每一个取值组合进行检验；
  - ⑤ 输出符合条件的取值组合作为问题的解。

【练习】周瑜的年龄

- 求解年龄
- 确定年龄的范围
  - 而立之年，早逝两位数
- 枚举所有年龄值
- 逐个检验是否符合条件
  - 十位比个位小3
  - 个位平方等于年龄
- 输出问题的解

大江东去浪淘尽，千百年风流人物。
而立之年督东吴，早逝英年两位数。
十位恰小个位三，个位平方与寿符。
哪位学子算得快，多少年华属周瑜？

【练习】周瑜的年龄

```
01.     for x in range(30,100):
02.         n = x / 10
03.         m = x % 10
04.         if m - n == 3 and m * m == x:
05.             print (x)
```

【练习】SEND MORE MONEY

- 求每个字母所代表的数字，让等式成立
- 确定问题解包含的变量
  - S、E、N、D、M、O、R、N、Y
- 每个变量的取值范围
  - range(10)
- 枚举变量值的所有组合
- 检验等式是否成立
- 输出答案

什么是算法分析

- 如何对比两个程序?
  - 看起来不同，但解决同一个问题的程序，哪个“更好”？
- 程序和算法的区别
  - 算法是对问题解决的分步描述
  - 程序则是采用某种编程语言实现的算法
- 同一个算法通过不同的程序员采用不同的编程语言，能产生很多程序

什么是算法分析

- 我们来看一段程序
- 完成从1到n的累加，输出总和
  - 设置累计变量=0
  - 从1到n循环，逐次累加到累计变量
  - 返回累计变量

```
def sumOfN(n):
    theSum = 0
    for i in range(1, n + 1):
        theSum = theSum + i
    return theSum

print(sumOfN(10))
```

什么是算法分析

- 再看第二段程序，是否感觉怪怪的?
  - 但实际上本程序功能与前面那段相同
- 这段程序失败之处在于:
  - 变量命名
  - 有无用的垃圾代码
- 比较程序的“好坏”，有更多因素
  - 代码风格、可读性等等

```
def foo(tom):
    fred = 0
    for bill in range(1, tom + 1):
        barney = bill
        fred = fred + barney
    return fred


print(foo(10))
```

说到代码风格和可读性

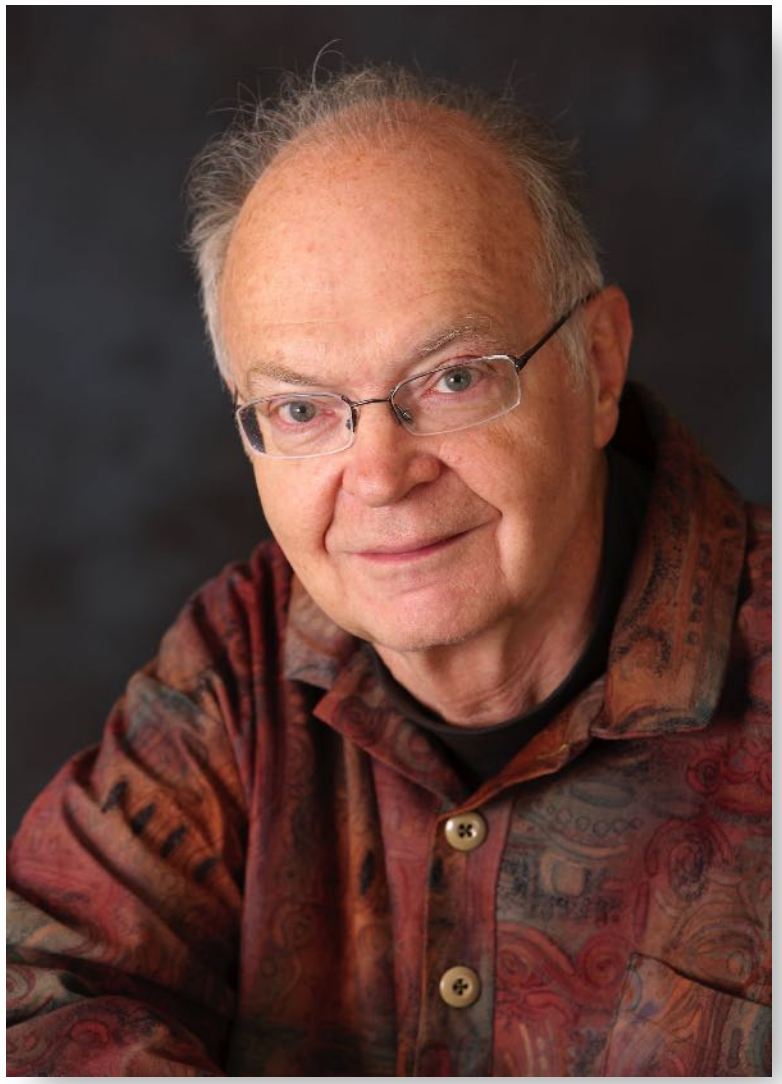
- 为什么Python的语句块强制缩进是好的?
  - 语句块功能和视觉效果统一
- 苹果公司的一个低级Bug
  - 造成SSL连接验证被跳过
  - 2014.2.22修正iOS7.0.6
- 不像看起来那样运行

```
7
8       if ((err = SSLHashSHA1
9           goto fail;
10      if ((err = SSLHashSHA1
11          goto fail;
12          goto fail;
13      if ((err = SSLHashSHA1
14          goto fail;
15      err = sslRawVerify(ctx
```

```
1  static OSStatus
2  SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedPa
3                                   uint8_t *signature, UInt16 signatureLen)
4  {
5      OSStatus        err;
6      ...
7
8      if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
9          goto fail;
10     if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11         goto fail;
12         goto fail;
13     if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14         goto fail;
15     err = sslRawVerify(ctx,
16                        ctx->peerPubKey,
17                        dataToSign,              /* plaintext */
18                        dataToSignLen,           /* plaintext length */
19                        signature,
20                        signatureLen);
21     if(err) {
22         sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
23                     "returned %d\n", (int)err);
24         goto fail;
25     }
26
27 fail:
28     SSLFreeBuffer(&signedHashes);
29     SSLFreeBuffer(&hashCtx);
30     return err;
31 }
```

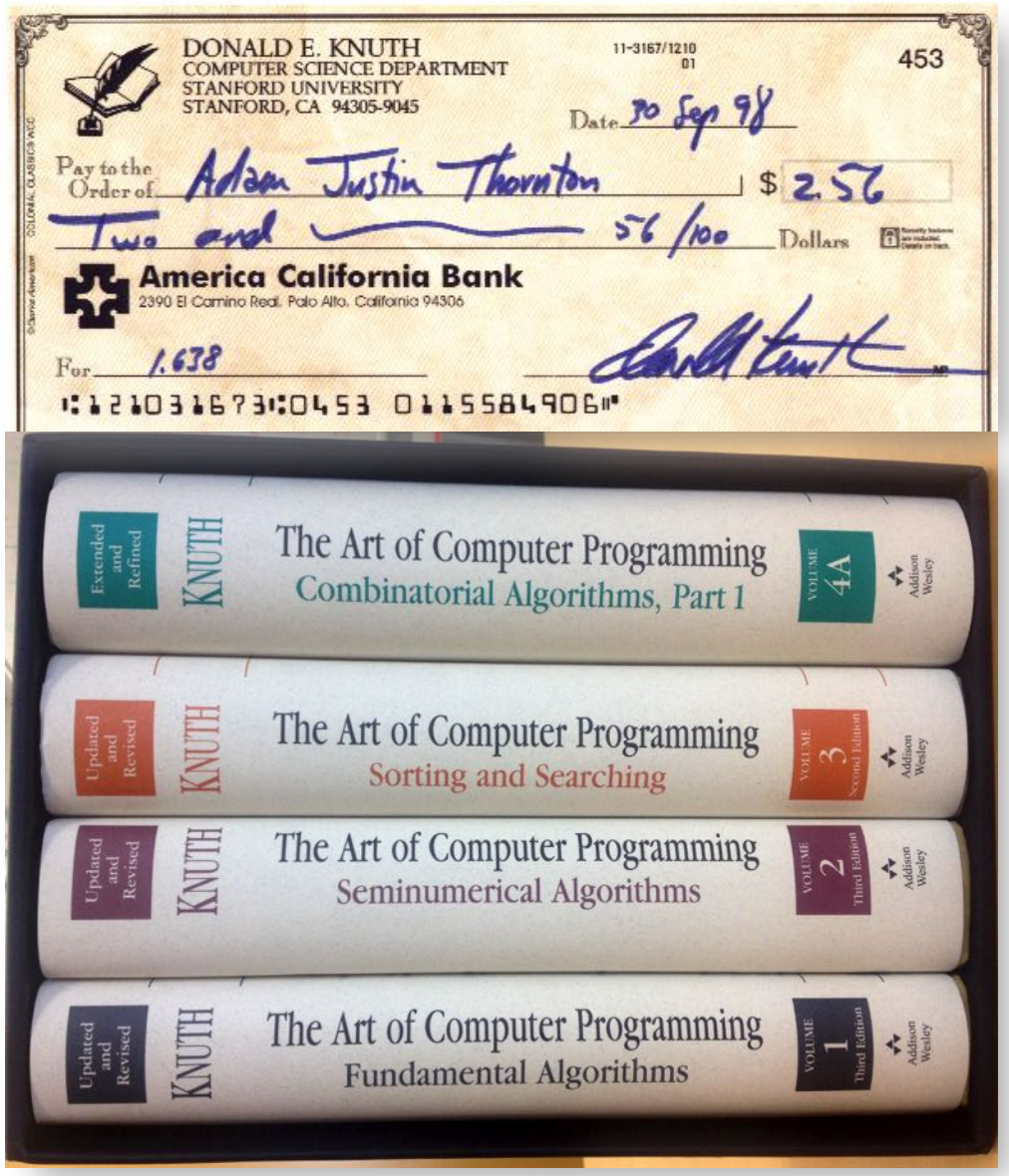
程序是写给人读的，只是偶尔让计算机执行一下

- Python的强制缩进规范完成了关键部分
- 我们还需要良好的编程规范
  - 变量、函数、类命名
  - 注释和文档
  - 一些编程设计上的良好风格
- Programs are meant to be read by humans and only incidentally for computers to execute.—Donald Ervin Knuth



程序员的精彩人生

- 鸿篇巨著《计算机程序设计艺术》
- 为了巨著印刷漂亮，开发了伟大的排版软件TEX
  - 有趣的版本号3.14159265，最后是π
  - 奖励bug提交者，从128美分开始翻倍
  - 但得到奖金的人往往不愿拿支票去兑现
- 字符串快速匹配KMP算法
- 1974年获得图灵奖



Python之禅（Tim Peters）

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Python之禅（赖勇浩翻译）

- 优美胜于丑陋（Python 以编写优美的代码为目标）
- 明了胜于晦涩（优美的代码应当是明了的，命名规范，风格相似）
- 简洁胜于复杂（优美的代码应当是简洁的，不要有复杂的内部实现）
- 复杂胜于凌乱（如果复杂不可避免，那代码间也不能有难懂的关系，要保持接口简洁）
- 扁平胜于嵌套（优美的代码应当是扁平的，不能有太多的嵌套）
- 间隔胜于紧凑（优美的代码有适当的间隔，不要奢望一行代码解决问题）
- 可读性很重要（优美的代码是可读的）
- 即便假借特例的实用性之名，也不可违背这些规则（这些规则至高无上）

Python之禅（赖勇浩翻译）

- 不要包容所有错误，除非你确定需要这样做（精准地捕获异常，不写except:pass 风格的代码）
- 当存在多种可能，不要尝试去猜测
- 而是尽量找一种，最好是唯一一种明显的解决方案（如果不确定，就用穷举法）
- 虽然这并不容易，因为你不是 Python 之父（这里的 Dutch 是指 Guido ）
- 做也许好过不做，但不假思索就动手还不如不做（动手之前要细思量）
- 如果你无法向人描述你的方案，那肯定不是一个好方案；反之亦然（方案测评标准）
- 命名空间是一种绝妙的理念，我们应当多加利用（倡导与号召）

什么是算法分析

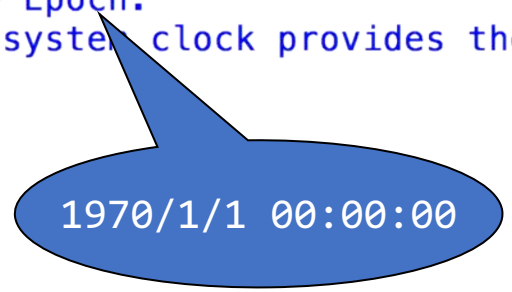
- 我们主要感兴趣的是算法本身特性
- 算法分析主要就是从计算资源消耗的角度来评判和比较算法
  - 更高效利用计算资源，或者更少占用资源的算法，就是好算法
  - 从这个角度，前述两段程序实际上是基本相同的
  - 它们都采用了一样的算法来解决累计求和问题

```
def sumOfN(n):
    theSum = 0
    for i in range(1, n + 1):
        theSum = theSum + i
    return theSum

print(sumOfN(10))
```

```
def foo(tom):
    fred = 0
    for bill in range(1, tom + 1):
        barney = bill
        fred = fred + barney
    return fred


print(foo(10))
```

计算资源指标

```
>>> help(time.time)
Help on built-in function time in module time:

time(...)
    time() -> floating point number

    Return the current time in seconds since the Epoch.
    Fractions of a second may be present if the system clock provides them.
```



- 那么何为计算资源?
- 一种是算法解决问题过程中需要的存储空间或内存
  - 但存储空间受到问题自身数据规模的变化影响
  - 要区分哪些存储空间是问题本身描述所需，哪些是算法占用，不容易
- 另一种是算法的执行时间
  - 我们可以对程序进行实际运行测试，获得真实的运行时间
- Python中有一个time模块，可以获取计算机系统当前时间
  - 在算法开始前和结束后分别记录系统时间，即可得到运行时间

```
>>> import time
>>> time.time()
1551764033.4839659
>>>
```

运行时间检测

- 累计求和程序的运行时间检测
  - 增加了总运行时间
  - 函数返回一个元组tuple
  - 包括累计和，以及运行时间（秒）
- 在交互窗口连续运行5次看看
  - 1到10,000累加
  - 每次运行约需0.0007秒

```
1   import time
2
3
4   def sumOfN2(n):
5       start = time.time()
6       theSum = 0
7       for i in range(1, n + 1):
8           theSum = theSum + i
9       end = time.time()
10      return theSum, end - start
11
12
13  for i in range(5):
14      print("Sum is %d required %10.7f seconds"
15            % sumOfN2(10000))
```

```
Sum is 50005000 required  0.0007980 seconds
Sum is 50005000 required  0.0007021 seconds
Sum is 50005000 required  0.0007031 seconds
Sum is 50005000 required  0.0007219 seconds
Sum is 50005000 required  0.0007060 seconds
```

运行时间检测

- 如果累加到100,000?
  - 看起来运行时间增加到10,000的10倍

```
13    for i in range(5):
14        print("Sum is %d required %10.7f seconds" % sumOfN2(100000))
```

```
Sum is 5000050000 required  0.0078530 seconds
Sum is 5000050000 required  0.0078511 seconds
Sum is 5000050000 required  0.0087960 seconds
Sum is 5000050000 required  0.0082700 seconds
Sum is 5000050000 required  0.0077040 seconds
```

- 进一步累加到1,000,000?
  - 运行时间又是100,000的10倍了

```
13    for i in range(5):
14        print("Sum is %d required %10.7f seconds" % sumOfN2(1000000))
```

```
Sum is 500000500000 required  0.0817859 seconds
Sum is 500000500000 required  0.0781529 seconds
Sum is 500000500000 required  0.0803380 seconds
Sum is 500000500000 required  0.0783160 seconds
Sum is 500000500000 required  0.0776238 seconds
```

第二种无迭代的累计算法

- 利用求和公式的无迭代算法
- 采用同样的方法检测运行时间
  - 10,000; 100,000; 1,000,000
  - 10,000,000; 100,000,000
- 需要关注的两点
  - 这种算法的运行时间比前种都短很多
  - 运行时间与累计对象n的大小没有关系（前种算法是倍数增长关系）
- 新算法运行时间几乎与需要累计的数目无关

```
17    def sumOfN3(n):
18        start = time.time()
19        theSum = (n * (n + 1)) / 2
20        end = time.time()
21        return theSum, end - start
```

```
Sum is 50005000 required  0.0000010 seconds
Sum is 5000050000 required  0.0000000 seconds
Sum is 500000500000 required  0.0000010 seconds
Sum is 50000005000000 required  0.0000000 seconds
Sum is 5000000050000000 required  0.0000169 seconds
```

运行时间检测的分析

- 观察一下第一种迭代算法
  - 包含了一个循环，可能会执行更多语句
  - 这个循环运行次数跟累加值n有关系，n增加，循环次数也增加
- 但关于运行时间的实际检测，有点问题
  - 关于编程语言和运行环境
- 同一个算法，采用不同的编程语言编写，放在不同的机器上运行，得到的运行时间会不一样，有时候会大不一样：
  - 比如把非迭代算法放在老旧机器上跑，甚至可能慢过新机器上的迭代算法
- 所以我们需要更好的方法来衡量算法的运行时间
  - 这个指标与具体的机器、程序、运行时段，与编译器都无关

大O表示法（Big-O）

- 一个算法所实施的操作数量或步骤数可作为独立于具体程序/机器的度量指标
  - 哪种操作跟算法的具体实现无关?
  - 需要一种通用的基本操作来作为运行步骤的计量单位
- 赋值语句是一个合适的选择
  - 一条赋值语句同时包含了（表达式）计算和（变量）存储两个基本资源
  - 仔细观察程序设计语言特性，除了与计算资源无关的定义语句之外，主要就是三种控制流语句和赋值语句，而控制流仅仅起了组织赋值语句的作用，并不实施处理。
- 分析SumOfN的赋值语句数量
  - 对于“问题规模”n
  - 赋值语句数量T(n)=1+n

```
31  def sumOfN(n):
32      theSum = 0
33      for i in range(1, n + 1):
34          theSum = theSum + i
35      return theSum
```

大O表示法（Big-O）

- 问题规模影响算法执行时间
  - 在前n个自然数累计求和的算法中，需要累计的自然数个数合适作为问题规模的指标
  - 前100,000个自然数求和对比前1,000个自然数求和，算是同一问题的更大规模
  - 算法分析的目标是要找出问题规模会怎么影响一个算法的执行时间
- 数量级函数（Order of Magnitude function）
  - 基本操作数量函数T(n)的精确值并不是特别重要，重要的是T(n)中起决定性因素的主导部分
  - 用动态的眼光看，就是当问题规模增大的时候，T(n)中的一些部分会盖过其它部分的贡献
  - 数量级函数描述了T(n)中随着n增加而增加速度最快的部分
  - 称作“大O”表示法，记作O(*f*(*n*))，其中*f*(*n*)表示T(n)中的主导部分

确定运行时间数量级大O的方法

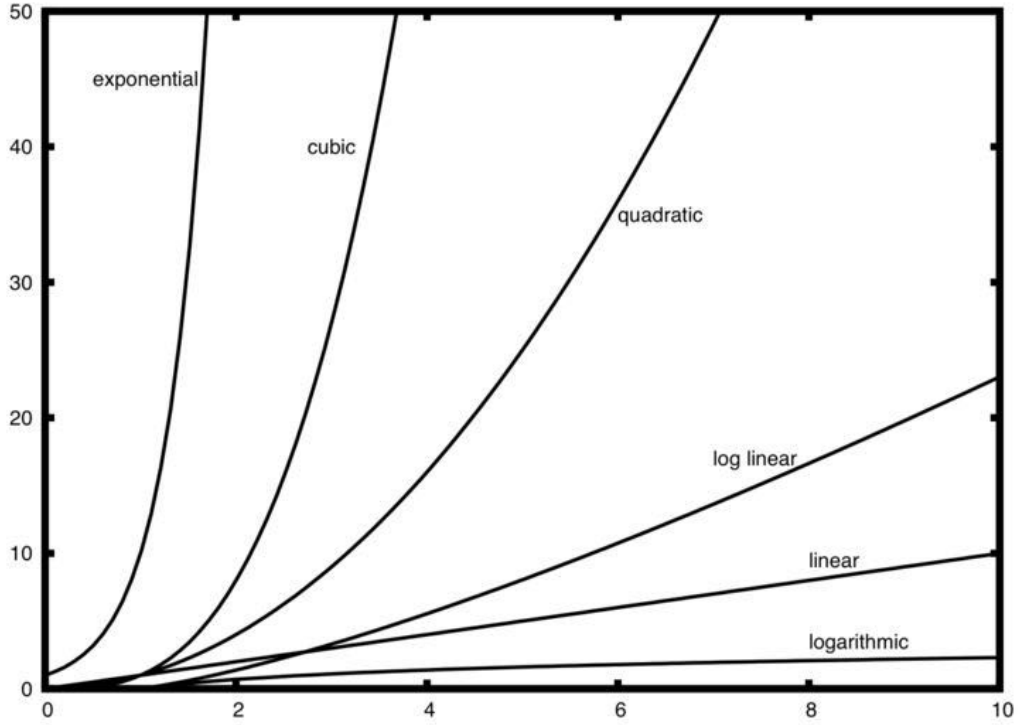
- 例1：T(n)=1+n
  - 当n增大时，常数1在最终结果中显得越来越无足轻重
  - 所以可以去掉1，保留n作为主要部分，运行时间数量级就是O(n)
- 例2：T(n)=5n^2+27n+1005
  - 当n很小时，常数1005其决定性作用
  - 但当n越来越大，n^2项就越来越重要，其它两项对结果的影响则越来越小
  - 同样，n^2项中的系数5，对于n^2的增长速度来说也影响不大
  - 所以可以在数量级中去掉27n+1005，以及系数5的部分，确定为O(n^2)

确定运行时间数量级大O的方法

- 有时决定运行时间的不仅是问题规模
- 某些具体数据也会影响算法运行时间
  - 分为最好、最差和平均情况，平均状况体现了算法的主流性能
  - 对算法的分析要看主流，而不被某几种特定的运行状况所迷惑

常见的大O数量级函数

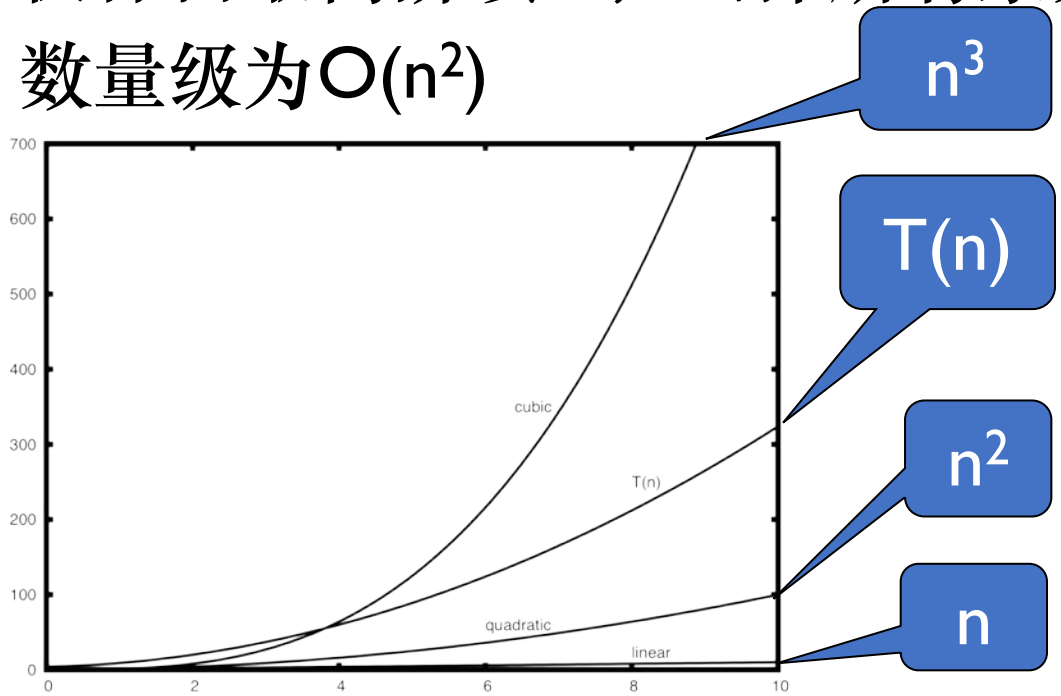
- 通常当n较小时，难以确定其数量级
- 当n增长到较大时，容易看出其主要变化量级



$f(n)$	名称
1	常数
$Log(n)$	对数
n	线性
$n*Log(n)$	对数线性
n^2	平方
n^3	立方
2^n	指数

从代码分析确定执行时间数量级函数

- 代码赋值语句可以分为4个部分
  - $T(n)=3+3n^2+2n+1=3n^2+2n+4$
- 仅保留最高阶项n^2，去掉所有系数
- 数量级为$O(n^2)$



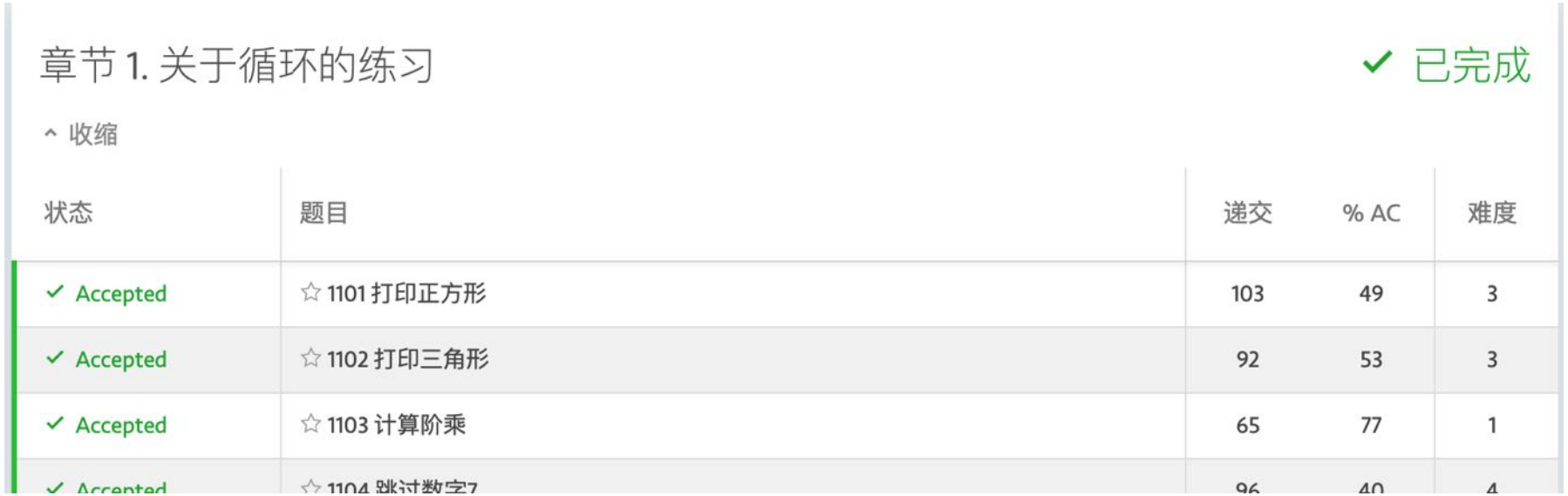

```
38    a = 5
39    b = 6
40    c = 10
41    for i in range(n):
42        for j in range(n):
43            x = i * i
44            y = j * j
45            z = i * j
46    for k in range(n):
47        w = a * k + 45
48        v = b * b
49    d = 33
```

随堂作业

- 写两个Python程序，功能为找到列表中最小数
  - 别用list.sort()
- 要求程序1将每个数与其它所有数比较，运行时间数量级为O(n^2)
- 要求程序2数量级为O(n)

【H6】完成OJ题：循环和条件分支

- 请到如下网址完成练习题：
  - https://vijos.org/d/pkuchenbin/training/5c96d79cf413620934d0a6e5
- 做完后将带有Accepted字样的截图上传提交表单



章节 1. 关于循环的练习 ✓ 已完成

收缩

状态	题目	递交	% AC	难度
✓ Accepted	☆ 1101 打印正方形	103	49	3
✓ Accepted	☆ 1102 打印三角形	92	53	3
✓ Accepted	☆ 1103 计算阶乘	65	77	1
✓ Accepted	☆ 1104 跳过数字7	96	40	4