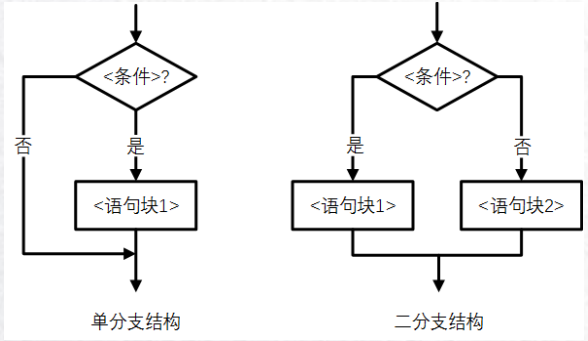
《人工智能与Python程序设计》——函数与代码复用

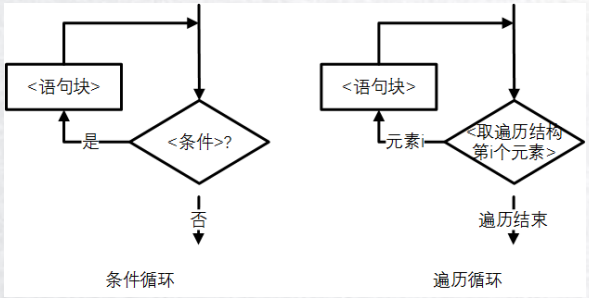
人工智能与Python程序设计 教研组

上次课回顾：程序的控制结构

- Python中的分支结构
  - 单分支
  - 二分支
  - 多分支
  - 异常处理



- Python中的循环结构
  - 遍历循环
  - 条件循环
  - break
  - continue



语句(Statement) vs. 表达式（Expression）

- 语句：
  - Python语言编写的程序是由注释和语句组成的
  - 赋值语句（=，+=，-=，....）
  - 分支、循环、异常处理
  - 函数定义（def）
  - 类定义(class)
  - ……
- 表达式：
  - 由变量名(identifiers)、常量(literals)、运算符（operators）、函数调用()、下标运算[]、括号等元素组成
  - 一个合法的表达式可以被**求值**：经过计算表达式会返回一个值
  - Python解释器在执行一条语句时，看到其中包含的表达式就会对它进行求值
  - 单独一个表达式也是一条语句
  - 注意：赋值语句不是一个表达式
- if 语句 vs. if 表达式

Python AI
函数与代码复用

提纲

- **函数的基本使用**
- 函数的参数传递
- 函数递归

函数的定义

- 函数是一段具有特定功能的、可重用的语句组，用函数名来表示并通过函数名完成功能调用。
  - 输入参数传递
  - 返回输出
- 函数也可以看作是一段具有**名字**的**子程序**，可以在需要的地方调用执行，不需要在每个执行地方重复编写这些语句。每次使用函数可以提供不同的参数作为输入，以实现对不同数据的处理；函数执行后，还可以反馈相应的处理结果。
- 一种**功能抽象**：黑盒

函数的定义

- 作用
  - 问题抽象和分解，降低编程难度
  - 代码复用
- Python自带函数
  - 内置函数
  - Python标准库中的函数

- 自定义函数：使用def保留字

```
def <函数名>(<参数列表>):
    <函数体>
    return <返回值列表>
```

示例：生日歌

- 生日歌：歌词
  - Happy birthday to you!
  - Happy birthday to you!
  - Happy birthday, dear <名字>!
  - Happy birthday to you!
- 编写程序为Mike和Lily输出生日歌歌词

```
In [1]: print("Happy birthday to you!")
        print("Happy birthday to you!")
        print("Happy birthday, dear Mike!")
        print("Happy birthday to you!")

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Mike!
Happy birthday to you!
```

示例：生日歌

- 定义函数HappyB()
  - 括号中<名字>形参
  - 调用两次

In [2]:

```
def happy():
    print("Happy birthday to you!")
def happyB(name):
    happy()
    happy()
    print("Happy birthday, dear {}!".format(name))
    happy()

happyB("Mike")
print()
happyB("Lily")
```

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Mike!
Happy birthday to you!

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Lily!
Happy birthday to you!
```

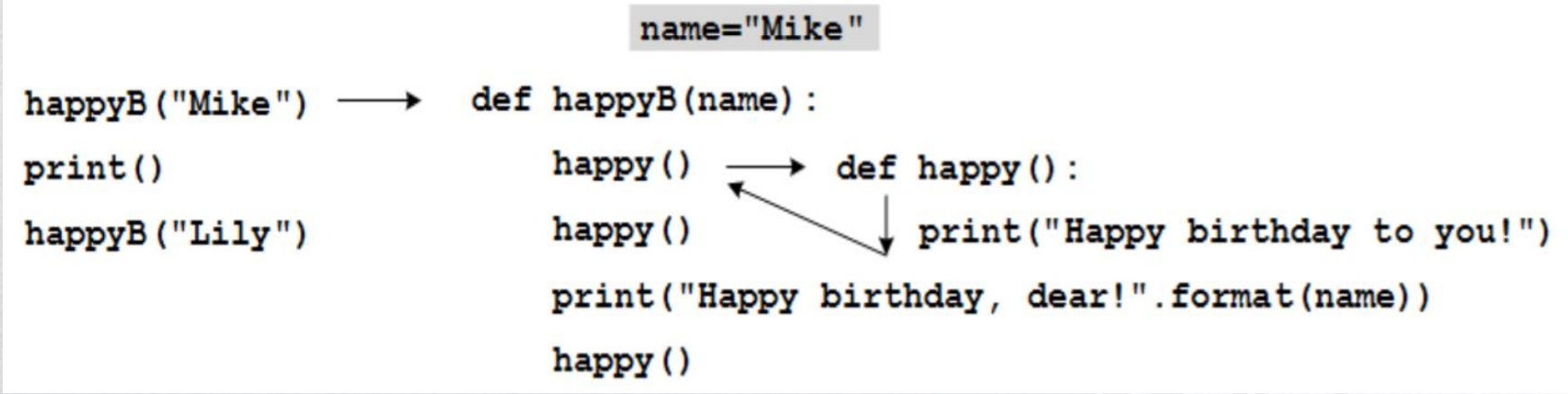
函数调用过程

- 四个步骤：
- （1）调用程序在调用处暂停执行
- （2）在调用时将实参传递（赋值！）给函数的形参
- （3）执行函数体语句
- （4）函数调用结束给出返回值，程序回到调用前的暂停处继续执行

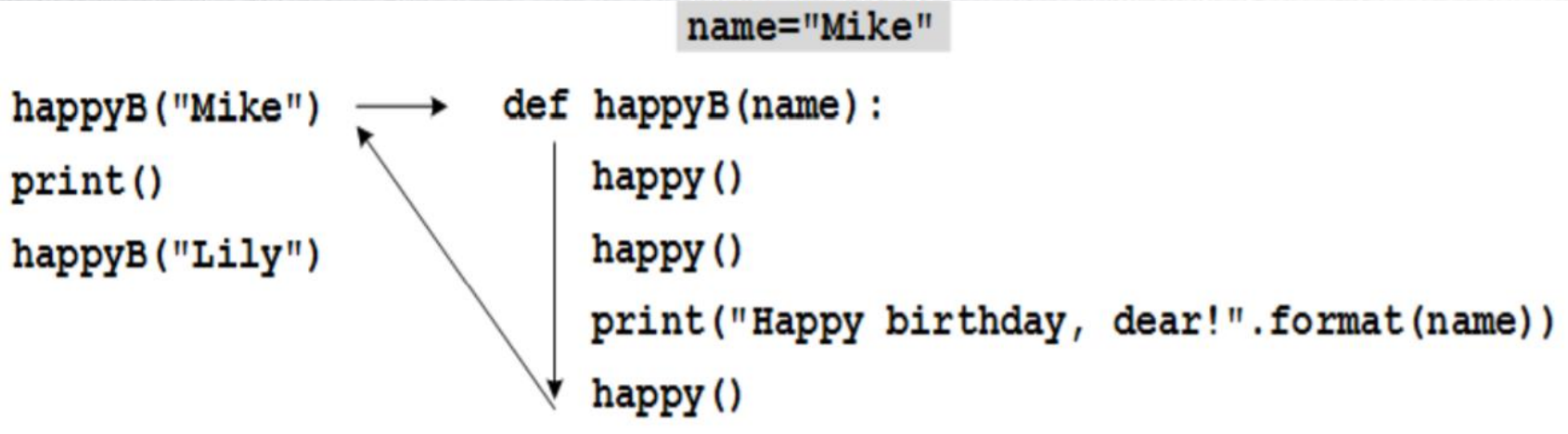
函数调用过程

```
                                    name="Mike"
happyB("Mike")  ──────►   def happyB(name):
print()                       happy()
happyB("Lily")                happy()
                              print("Happy birthday, dear!".format(name))
                              happy()
```

函数调用过程


name="Mike"
happyB("Mike")
print()
happyB("Lily")
def happyB(name):
happy()
happy()
print("Happy birthday, dear!".format(name))
happy()
def happy():
print("Happy birthday to you!")

函数调用过程


name="Mike"
happyB("Mike")
print()
happyB("Lily")
def happyB(name):
happy()
happy()
print("Happy birthday, dear!".format(name))
happy()

lambda函数

- lambda保留字：用来定义一种特殊的函数——匿名函数，又称lambda函数。

- 通常会将新创建的匿名函数赋值给一个变量：
  - <函数名> = lambda <参数列表>: <表达式>
  - *<函数名>是一个类型为函数的变量*

- lambda函数与正常函数一样，等价于下面形式

```
def <函数名>(<参数列表>):
    return <表达式>
```

lambda函数

- lambda函数用于定义简单的、能够在一行内表示的函数，返回一个函数类型。

```
In  [4]: f = lambda x, y : x+y
         type(f)

Out[4]: function

In  [5]: f(10,12)

Out[5]: 22
```

单选题 1分

设置

下列Python程序定义f1()时还没有定义f2()，这种函数调用是否合法?

```
In [8]: def f1():
            f2()
        def f2():
            print("This is function f2()")
        f1()
```

A 合法

B 不合法

提交

Python AI
函数与代码复用

提纲

- 函数的基本使用
- **函数的参数传递**
- 函数递归

可选参数

- 在定义函数时，有些参数可以存在默认值
- 函数被调用时，如果没有传入对应的参数值，则使用函数定义时的默认值代替
- 可选参数必须定义在非可选参数后面

```
In [6]: def dup(str, times=2):
            print(str*times)
        dup("ruc~")

ruc~ruc~

In [7]: dup("ruc!",4)

ruc!ruc!ruc!ruc!
```

可变数量参数

- 在函数定义时，可以设计可变数量参数，通过参数前增加星号（*）实现
- 可变参数只能出现在参数列表的最后
  - 可变参数被当作元组类型传入函数中

```
In [9]: def vfunc(a, *b):
            print(type(b))
            for n in b:
                a += n
            return a
        vfunc(1,2,3,4,5)
```

```
<class 'tuple'>
```

Out[9]: 15

参数的位置和名称传递

- 实参默认采用按照位置顺序的方式传递给函数
  - *def func(x1,y1,z1,x2,y2,z2)*
  - *func(1,2,3,4,5,6)*

- Python提供了按照形参名称输入实参的方式，调用如下：
  - *result = func(x2=4, y2=5, z2=6, x1=1, y1=2, z1=3)*

- 由于调用函数时指定了参数名称，所以参数之间的顺序可以任意调整

函数的返回值

- return语句用来退出函数并将程序返回到函数被调用的位置继续执行
- return语句同时可以将0个、1个或多个函数运算完的结果返回给函数被调用处的变量
  - 用return返回多个值，多个值以元组类型保存
- 函数可以没有return，此时函数返回None

```
In [10]: def func(a,b):
             return a*b
         s = func("ruc!",2)
         print(s)

         ruc!ruc!
```

```
In [11]: def func(a,b):
             return b, a
         s = func("ruc!",2)
         print(s, type(s))

         (2, 'ruc!') <class 'tuple'>
```

函数对变量的作用

- 一个程序中的变量包括两类：全局变量和局部变量
  - 全局变量指在函数之外定义的变量，一般没有缩进，在程序执行全过程有效
  - 局部变量指在函数内部使用的变量，仅在函数内部有效当函数退出时变量将不存在

In [12]:
```
n = 1 #n是全局变量
def func(a, b):
    c = a * b #c是局部变量，a和b作为函数参数也是局部变量
    return c
s = func("ruc!", 2)
print(c)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-07b21183da77> in <module>
      4     return c
      5 s = func("ruc!", 2)
----> 6 print(c)

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

函数对变量的作用

- 函数内部使用全局变量

In [13]:
```
n = 1 #n是全局变量
def func(a, b):
    n = b #这个n是在函数内存中新生成的局部变量，不是全局变量
    return a*b
s = func("ruc!", 2)
print(s, n) #测试一下n值是否改变
```

```
ruc!ruc! 1
```

- 这里n被函数理解为一个局部变量，函数退出后释放

函数对变量的作用

- 如果希望让func()函数将n当作全局变量，需要在变量n使用前显式声明该变量为全局变量

In [14]:
```
n = 1 #n是全局变量
def func(a, b):
    global n
    n = b #将局部变量b赋值给全局变量n
    return a*b
s = func("ruc!", 2)
print(s, n) #测试一下n值是否改变
```

```
ruc!ruc! 2
```

函数对变量的作用

- 如果全局变量的类型为组合数据类型，比如列表类型：

In [15]:
```
ls = [] #ls是全局列表变量
def func(a, b):
    ls.append(b) #将局部变量b增加到全局列表变量ls中
    return a*b
s = func("ruc!", 2)
print(s, ls) #测试一下ls值是否改变
```

```
ruc!ruc! [2]
```

- 全局列表变量在函数调用后发生变化

函数对变量的作用

- 如果func()函数内部存在一个真实创建过且名称为ls的列表，则func()将操作该列表而不会修改全局变量

In [16]:
```
ls = [] #ls是全局列表变量
def func(a, b):
    ls = [] #创建了名称为ls的局部列表变量列
    ls.append(b) #将局部变量b增加到全局列表变量ls中
    return a*b
s = func("ruc!", 3)
print(s, ls) #测试一下ls值是否改变
```

```
ruc!ruc!ruc! []
```

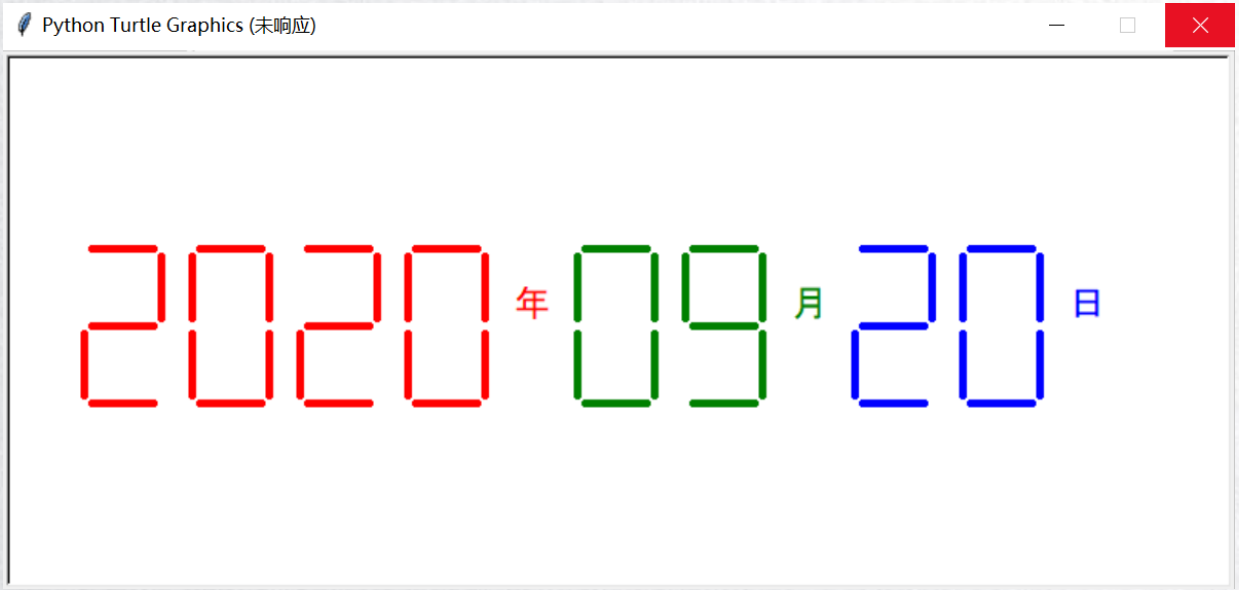
函数对变量的作用

- Python解释器是如何在函数内找到变量对应的值的？
  - 首先在局部变量中查找
  - 若找不到，则到上一级代码块中查找
    - 最后会找到在没有缩进的代码块中定义的全局变量
  - 若在函数内对变量进行赋值操作，则默认该变量是一个局部变量
  - 使用global语句，可以改变赋值操作的默认行为，不生成局部变量
    - 这样在函数内部就可以通过赋值语句更改全局变量的值了
  - 产生新问题：一个变量可以既是全局变量又是局部变量？
    - Python禁止了这种行为，抛出UnboundLocalError异常

```
In [2]: n = 1 #n是全局变量
        def func(a, b):
            print(n) #在局部变量中找不到n，所以在上层全局变量中查找n
            return a*b
        s = func("ruc!", 2)
        print(s, n) #测试一下n值是否改变

        1
        ruc!ruc! 1
```

```
In [3]: n = 1 #n是全局变量
        def func(a, b):
            print(n) #在局部变量中找不到n，所以在上层全局变量中查找n
            n = b #n是在函数内存中新生成的局部变量
            return a*b
        s = func("ruc!", 2)
        print(s, n) #测试一下n值是否改变

        ---------------------------------------------------------------------------
        UnboundLocalError                         Traceback (most recent call last)
        <ipython-input-3-7dc6bdb48cff> in <module>
              4     n = b #n是在函数内存中新生成的局部变量
              5     return a*b
        ----> 6 s = func("ruc!", 2)
              7 print(s, n) #测试一下n值是否改变

        <ipython-input-3-7dc6bdb48cff> in func(a, b)
              1 n = 1 #n是全局变量
              2 def func(a, b):
        ----> 3     print(n) #在局部变量中找不到n，所以在上层全局变量中查找n
              4     n = b #n是在函数内存中新生成的局部变量
              5     return a*b

        UnboundLocalError: local variable 'n' referenced before assignment
```

示例：数码管绘制



datetime库概述

- 以不同格式显示日期和时间是程序中最常用到的功能。Python提供了一个处理时间的标准函数库datetime，它提供了一系列由简单到复杂的时间处理方法。datetime库可以从系统中获得时间，并以用户选择的格式输出。
- datetime库以类的方式提供多种日期和时间表达方式：
  - datetime.date：日期表示类，可以表示年、月、日等
  - datetime.time：时间表示类，可以表示小时、分钟、秒、毫秒等
  - datetime.datetime：日期和时间表示的类，功能覆盖date和time类
  - datetime.timedelta：时间间隔有关的类
  - datetime.tzinfo：与时区有关的信息表示类

datetime库解析

from datetime import datetime

- 使用datetime.now()获得当前日期和时间对象，使用方法如下：

  datetime.now()

  – 作用：返回一个datetime类型，表示当前的日期和时间，精确到微秒

```
In [17]: from datetime import datetime
         today = datetime.now()
         today

Out[17]: datetime.datetime(2020, 9, 20, 13, 11, 27, 481215)
```

datetime库解析

- 使用datetime.utcnow()获得当前日期和时间对应的UTC（世界标准时间）时间对象，使用方法如下：

  datetime.utcnow()

  - 作用：返回datetime类型，表示当前日期和时间的UTC表示，精确到微秒。

```
In [18]: today = datetime.utcnow()
         today

Out[18]: datetime.datetime(2020, 9, 20, 5, 11, 57, 50987)
```

datetime库解析

- datetime.now() 和 datetime.utcnow() 都返回一个datetime类型的对象，也可以直接使用datetime()构造一个日期和时间对象：
  - datetime(year, month, day, hour=0, minute=0,second=0,microsecond=0)
  - 作用：返回一个datetime类型，表示指定的日期和时间，可以精确到微秒。

```
In [23]: someday = datetime(2020, 9, 20, 13, 12, 32, 7)
         someday

Out[23]: datetime.datetime(2020, 9, 20, 13, 12, 32, 7)
```

- 程序已经有了一个datetime对象，进一步可以利用这个对象的属性显示时间，为了区别datetime库名，采用上例中的someday代替生成的datetime对象

datetime库解析

属性	描述
someday.min	固定返回datetime的最小时间对象，datetime(1,1,1,0,0)
someday.max	固定返回datetime的最大时间对象，datetime(9999, 12, 31, 23, 59, 59, 999999)
someday.year	返回someday包含的年份
someday.month	返回someday包含的月份
someday.day	返回someday包含的日期
someday.hour	返回someday包含的小时
someday.minute	返回someday包含的分钟
someday.second	返回someday包含的秒钟
someday.microsecond	返回someday包含的微秒值

datetime库解析

- datetime对象有3个常用的时间格式化方法

属性	描述
someday.isoformat()	采用ISO 8601标准显示时间
someday.isoweekday()	根据日期计算星期后返回1-7,对应星期一到星期日
someday.strftime(format)	根据格式化字符串format进行格式显示的方法

- isoformat()和isoweekday()方法

```
In  [24]: someday.isoformat()
Out[24]: '2020-09-20T13:12:32.000007'
```

```
In  [25]: someday.isoweekday()
Out[25]: 7
```

- strftime()方法是时间格式化最有效的方法，几乎可以以任何通用格式输出时间

datetime库解析

- strftime()格式化控制符

格式化字符串	日期/时间	值范围和实例
%Y	年份	0001~9999，例如：1900
%m	月份	01~12，例如：10
%B	月名	January~December，例如：April
%b	月名缩写	Jan~Dec，例如：Apr
%d	日期	01～31，例如：25
%A	星期	Monday~Sunday，例如：Wednesday
%a	星期缩写	Mon~Sun，例如：Wed
%H	小时（24h制）	00～23，例如：12
%I	小时（12h制）	01～12，例如：7
%p	上/下午	AM, PM，例如：PM
%M	分钟	00～59，例如：26
%S	秒	00～59，例如：26

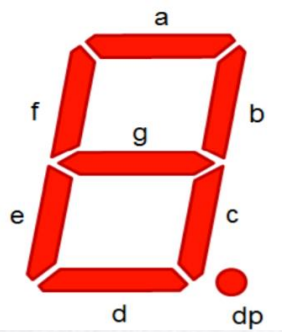
datetime库解析

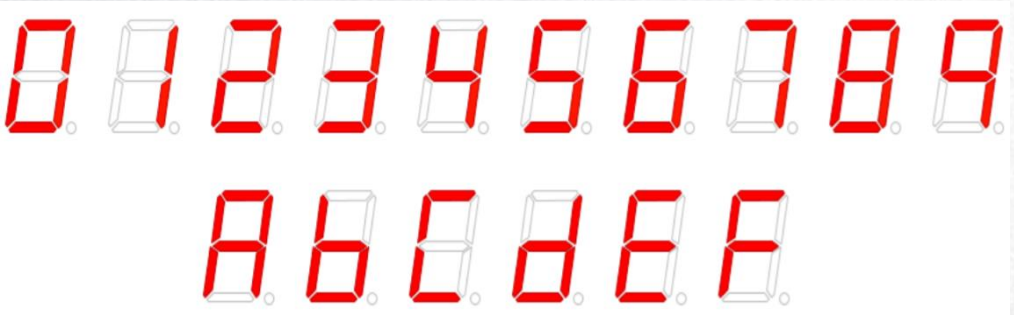
- strftime()格式化字符串的数字左侧会自动补零，上述格式也可以与print()的格式化函数一起使用

```
In  [26]: someday.strftime("%Y-%m-%d %H:%M:%S")
Out[26]: '2020-09-20 13:12:32'

In  [28]: someday.strftime("%A, %d. %B %Y %I:%M%p")
Out[28]: 'Sunday, 20. September 2020 01:12PM'

In  [29]: print("今天是{0:%Y}年{0:%m}月{0:%d}日".format(someday))
今天是2020年09月20日
```

示例：数码管绘制

- 七段数码管（seven-segment indicator）由7段数码管拼接而成，每段有亮或不亮两种情况，改进型的七段数码管还包括一个小数点位置
- 七段数码管能形成2^7=128种不同状态，其中部分状态能够显示易于人们理解的数字或字母含义，因此被广泛使用





示例：数码管绘制

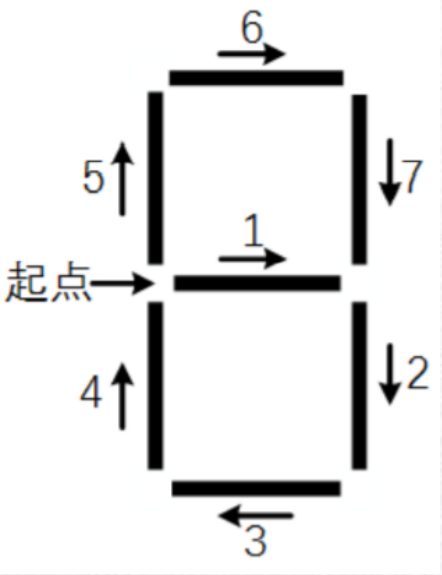
- 每个0到9的数字都有相同的七段数码管样式，因此，可以通过设计函数复用数字的绘制过程。进一步，每个七段数码管包括7个数码管样式，除了数码管位置不同外，绘制风格一致，也可以通过函数复用单个数码段的绘制过程。

In [32]:
```
import turtle, datetime

def drawLine(draw): #绘制单段数码管
    turtle.pendown() if draw else turtle.penup()
    turtle.fd(40)
    turtle.right(90)
```

示例：数码管绘制

- 实例代码定义了drawDigit()函数，该函数根据输入的数字d绘制七段数码管，结合七段数码管结构，每个数码管的绘制采用图所示顺序。

```
def drawDigit(d): #根据数字绘制七段数码管
    drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
    drawLine(True) if d in [0,1,3,4,5,6,7,8,9] else drawLine(False)
    drawLine(True) if d in [0,2,3,5,6,8,9] else drawLine(False)
    drawLine(True) if d in [0,2,6,8] else drawLine(False)
    turtle.left(90)
    drawLine(True) if d in [0,4,5,6,8,9] else drawLine(False)
    drawLine(True) if d in [0,2,3,5,6,7,8,9] else drawLine(False)
    drawLine(True) if d in [0,1,2,3,4,7,8,9] else drawLine(False)
    turtle.left(180)
    turtle.penup()
    turtle.fd(20)
```



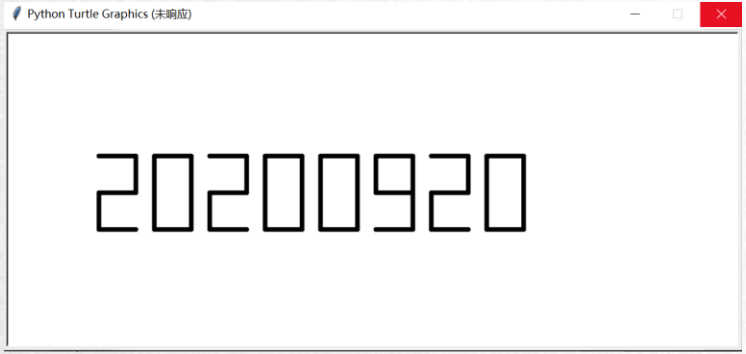
示例：数码管绘制

- 绘制起点在数码管中部左侧，无论每段数码管是否被绘制出来，turtle画笔都按顺序“画完”所有7个数码管。对于给定数字d，哪个数码段被绘制出来采用if...else...语句判断。

```
def drawDigit(d): #根据数字绘制七段数码管
    drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
    drawLine(True) if d in [0,1,3,4,5,6,7,8,9] else drawLine(False)
    drawLine(True) if d in [0,2,3,5,6,8,9] else drawLine(False)
    drawLine(True) if d in [0,2,6,8] else drawLine(False)
    turtle.left(90)
    drawLine(True) if d in [0,4,5,6,8,9] else drawLine(False)
    drawLine(True) if d in [0,2,3,5,6,7,8,9] else drawLine(False)
    drawLine(True) if d in [0,1,2,3,4,7,8,9] else drawLine(False)
    turtle.left(180)
    turtle.penup()
    turtle.fd(20)
```

示例：数码管绘制

```
def drawDate(date): #获得要输出的数字
    for i in date:
        drawDigit(eval(i)) #注意：通过eval()函数将数字变为整数

def main():
    turtle.setup(800, 350, 200, 200)
    turtle.penup()
    turtle.fd(-300)
    turtle.pensize(5)
    drawDate(datetime.datetime.now().strftime('%Y%m%d'))
    turtle.hideturtle()

main()
```

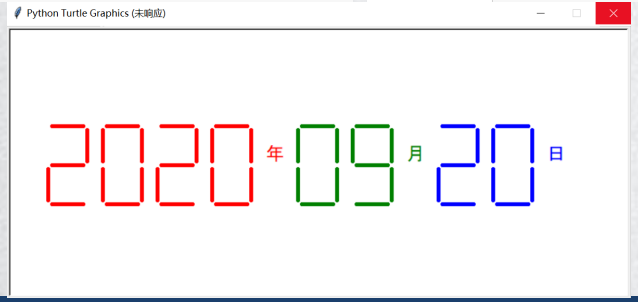


示例：数码管绘制

In [1]:

```
import turtle, datetime

def drawGap(): #绘制数码管间隔
    turtle.penup()
    turtle.fd(5)

def drawLine(draw): #绘制单段数码管
    drawGap()
    turtle.pendown() if draw else turtle.penup()
    turtle.fd(40)
    drawGap()
    turtle.right(90)
def drawDigit(d): #根据数字绘制七段数码管
    drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
    drawLine(True) if d in [0,1,3,4,5,6,7,8,9] else drawLine(False)
    drawLine(True) if d in [0,2,3,5,6,8,9] else drawLine(False)
    drawLine(True) if d in [0,2,6,8] else drawLine(False)
    turtle.left(90)
    drawLine(True) if d in [0,4,5,6,8,9] else drawLine(False)
    drawLine(True) if d in [0,2,3,5,6,7,8,9] else drawLine(False)
    drawLine(True) if d in [0,1,2,3,4,7,8,9] else drawLine(False)
    turtle.left(180)
    turtle.penup()
    turtle.fd(20)
```

```
def drawDate(date):
    turtle.pencolor("red")
    for i in date:
        if i == '-':
            turtle.write('年',font=("Arial", 18, "normal"))
            turtle.pencolor("green")
            turtle.fd(40)
        elif i == '=':
            turtle.write('月',font=("Arial", 18, "normal"))
            turtle.pencolor("blue")
            turtle.fd(40)
        elif i == '+':
            turtle.write('日',font=("Arial", 18, "normal"))
        else:
            drawDigit(eval(i))

def main():
    turtle.setup(800, 350, 200, 200)
    turtle.penup()
    turtle.fd(-350)
    turtle.pensize(5)
    drawDate(datetime.datetime.now().strftime('%Y-%m=%d+'))
    turtle.hideturtle()
main()
```



代码复用与模块化设计

- 函数是程序的一种基本**抽象方式**，它将一系列代码组织起来通过命名供其他程序使用。函数封装的直接好处是代码复用，任何其他代码只要输入参数即可调用函数，从而避免相同功能代码在被调用处重复编写。代码复用产生了另一个好处，当更新函数功能时，所有被调用处的功能都被更新。
- 当程序的长度在百行以上，如果不划分模块就算是最好的程序员也很难理解程序含义程序的可读性就已经很糟糕了。解决这一问题的最好方法是将一个程序分割成短小的程序段，每一段程序完成一个小的功能。无论面向过程和面向对象编程，对程序合理划分功能模块并基于模块设计程序是一种常用方法，被称为“模块化设计”。

代码复用与模块化设计

- 模块化设计一般有两个基本要求：
  - 紧耦合：尽可能合理划分功能块，功能块内部耦合紧密；
  - 松耦合：模块间关系尽可能简单，功能块之间耦合度低。

- 使用函数只是模块化设计的必要非充分条件，根据计算需求合理划分函数十分重要。一般来说，完成特定功能或被经常复用的一组语句应该采用函数来封装，并尽可能减少函数间参数和返回值的数量。

Python AI
函数与代码复用

提纲

- 函数的基本使用
- 函数的参数传递
- **函数递归**

递归的定义

- 函数作为一种代码封装，可以被其他程序调用，当然，也可以被函数内部代码调用。这种函数定义中调用函数自身的方式称为递归。
- 递归在数学和计算机应用上非常强大，能够非常简洁的解决重要问题。
- 数学上有个经典的递归例子叫阶乘，阶乘通常定义为：

$$n! = n(n-1)(n-2)...(1)$$

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & otherwise \end{cases}$$

- 递归的两个关键特征：
  - 存在一个或多个基例，基例不需要再次递归，它是确定的表达式；
  - 所有递归链要以一个或多个基例结尾。

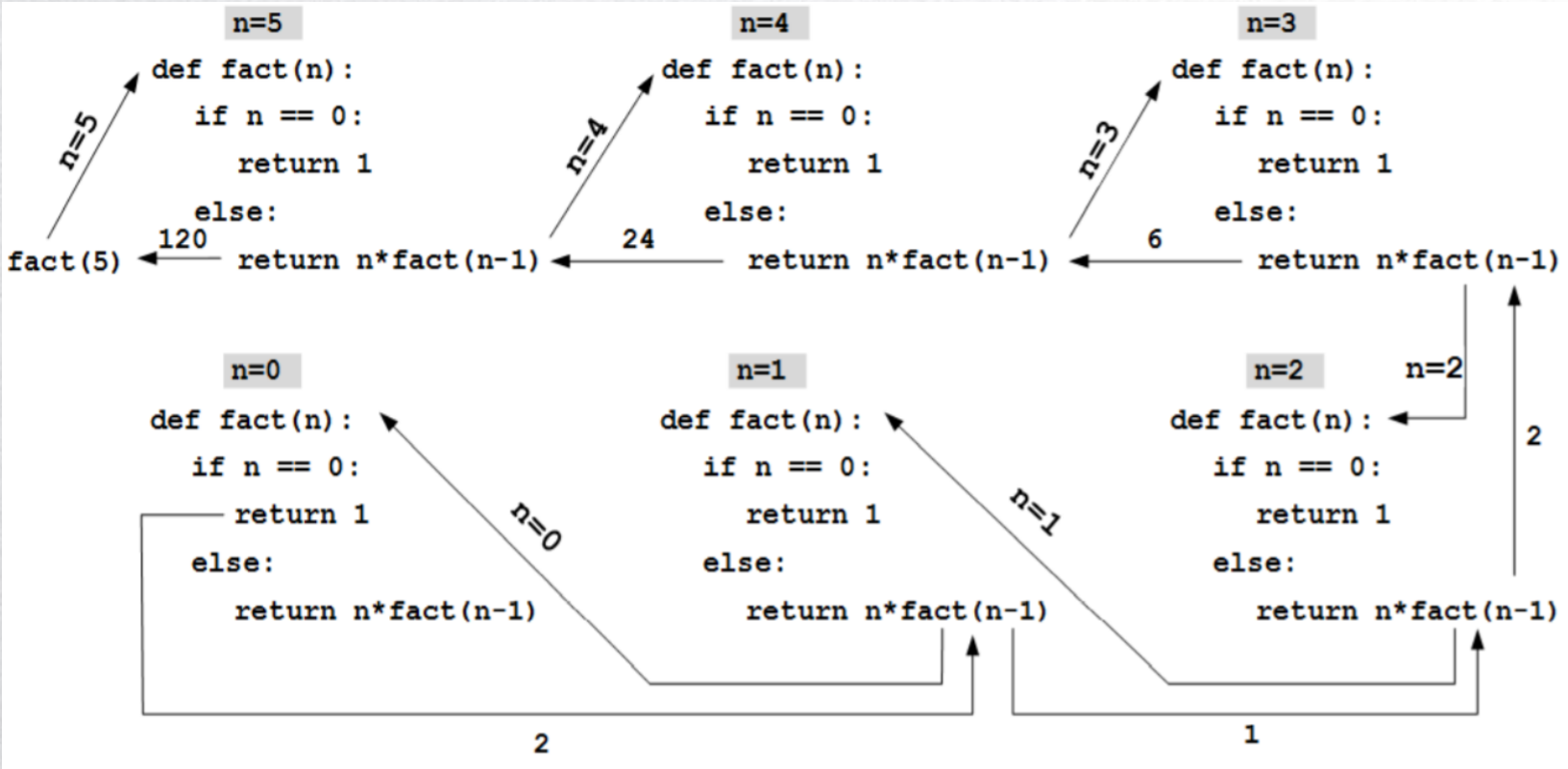
递归的使用方法

- 阶乘计算：用户输入整数n，计算并输出n的阶乘值

In [2]:
```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)
num = eval(input("请输入一个整数: "))
print(fact(abs(int(num))))
```

```
请输入一个整数: 10
3628800
```

递归的使用方法


n=5
def fact(n):
if n == 0:
return 1
else:
return n*fact(n-1)
n=5
fact(5)
120
n=4
def fact(n):
if n == 0:
return 1
else:
return n*fact(n-1)
n=4
24
n=3
def fact(n):
if n == 0:
return 1
else:
return n*fact(n-1)
n=3
6
n=2
def fact(n):
if n == 0:
return 1
else:
return n*fact(n-1)
n=2
2
n=1
def fact(n):
if n == 0:
return 1
else:
return n*fact(n-1)
n=1
1
n=0
def fact(n):
if n == 0:
return 1
else:
return n*fact(n-1)
n=0
2

递归的使用方法

- 字符串反转：对于用户输入的字符串s，输出反转后的字符串。
- 解决这个问题的基本思想是把字符串看作一个递归对象。

```
In [3]: def reverse(s):
            return reverse(s[1:]) + s[0]
        reverse("ABC")
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-3-9f2c1408d503> in <module>
      1 def reverse(s):
      2     return reverse(s[1:]) + s[0]
----> 3 reverse("ABC")

<ipython-input-3-9f2c1408d503> in reverse(s)
      1 def reverse(s):
----> 2     return reverse(s[1:]) + s[0]
      3 reverse("ABC")

... last 1 frames repeated, from the frame below ...

<ipython-input-3-9f2c1408d503> in reverse(s)
      1 def reverse(s):
----> 2     return reverse(s[1:]) + s[0]
      3 reverse("ABC")

RecursionError: maximum recursion depth exceeded
```

- 原因：reverse函数没有基例
- 为防止无限递归，Python设置了默认的最大递归深度

递归的使用方法

- 完整代码

In [1]:
```
def reverse(s):
    if s=="":
        return s
    else:
        return reverse(s[1:]) + s[0]

str = input("请输入一个字符串：")
print(reverse(str))
```

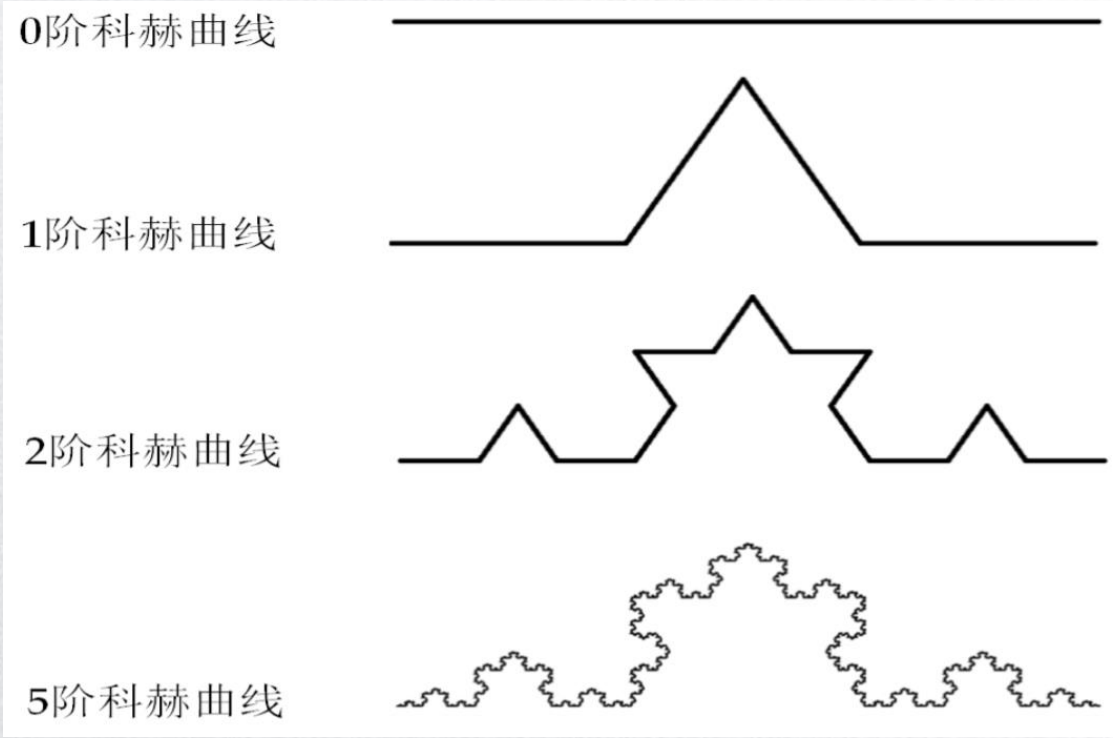
```
请输入一个字符串：高瓴人工智能学院
院学能智工人瓴高
```

科赫曲线绘制

- 自然界有很多图形很规则，符合一定的数学规律，例如，蜜蜂蜂窝是天然的等边六角形等。
- 科赫(Koch)曲线在众多经典数学曲线中非常著名，由瑞典数学家冯·科赫(H·V·Koch)于1904年提出，由于其形状类似雪花，也被称为雪花曲线。

科赫曲线绘制

正整数n代表科赫曲线的阶数，表示生成科赫曲线过程的操作次数。科赫曲线初始化阶数为0，表示一个长度为L的直线。对于直线L，将其等分为三段，中间一段用边长为L/3的等边三角形的两个边替代，得到1阶科赫曲线，它包含四条线段。进一步对每条线段重复同样的操作后得到2阶科赫曲线。继续重复同样的操作n次可以得到n阶科赫曲线。

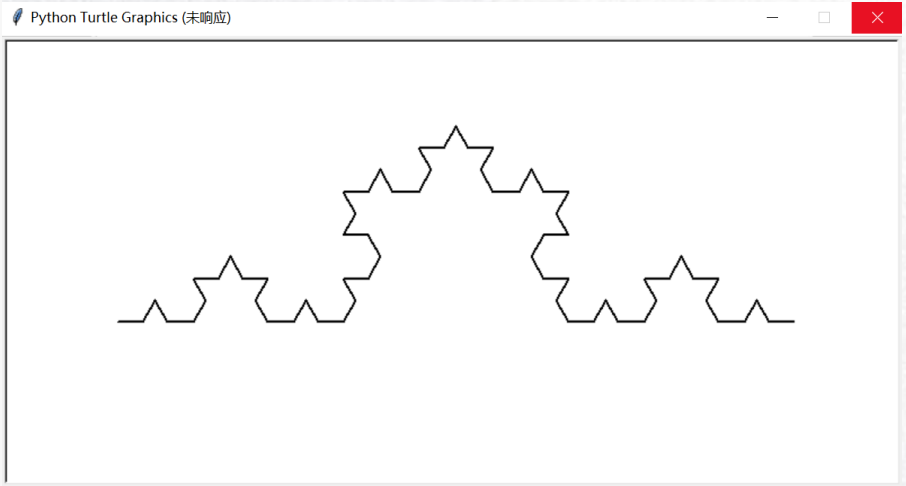


科赫曲线绘制

- 科赫曲线属于分形几 何分支，它的绘制过程 体现了递归思想，绘制过程代码。

In [2]:

```
import turtle
def koch(size, n):
    if n == 0:
        turtle.fd(size)
    else:
        for angle in [0, 60, -120, 60]:
            turtle.left(angle)
            koch(size/3, n-1)
def main():
    turtle.setup(800,400)
    turtle.speed(0) #控制绘制速度
    turtle.penup()
    turtle.goto(-300, -50)
    turtle.pendown()
    turtle.pensize(2)
    koch(600,3) # 科赫曲线长度，阶数
    turtle.hideturtle()
main()
```

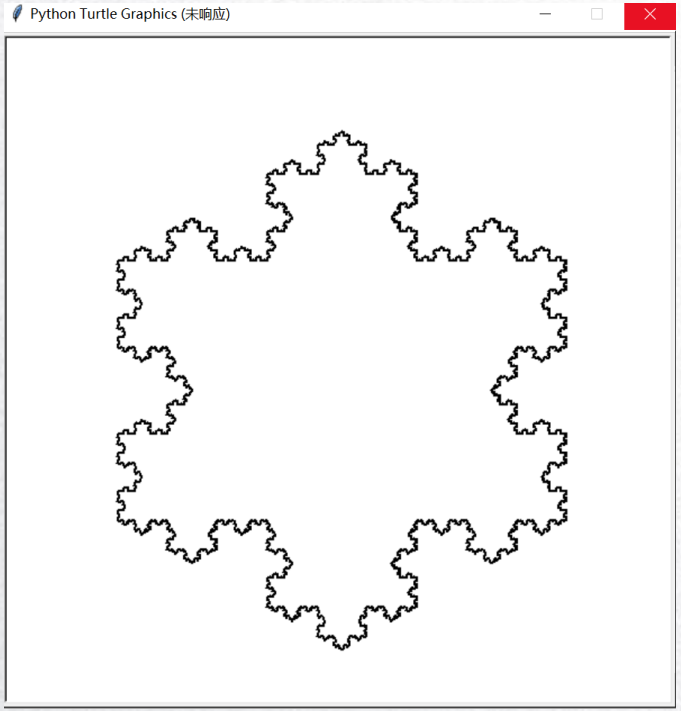


科赫曲线绘制

- 科赫曲线属于分形几 何分支，它的绘制过程 体现了递归思想，绘制过程代码。

In [1]:

```
import turtle
def koch(size, n):
    if n == 0:
        turtle.fd(size)
    else:
        for angle in [0, 60, -120, 60]:
            turtle.left(angle)
            koch(size/3, n-1)

def main():
    turtle.setup(600,600)
    turtle.speed(0)
    turtle.penup()
    turtle.goto(-200, 100)
    turtle.pendown()
    turtle.pensize(2)
    level = 5
    koch(400,level)
    turtle.right(120)
    koch(400,level)
    turtle.right(120)
    koch(400,level)
    turtle.hideturtle()

main()
```



Python内置函数

- Python解释器提供了68个内置函数

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

课后练习1（请在未来课堂系统提交）

- 用莱布尼茨级数法求圆周率，公式如下：
- 对比和蒙特卡洛法求圆周率在精度上的区别

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

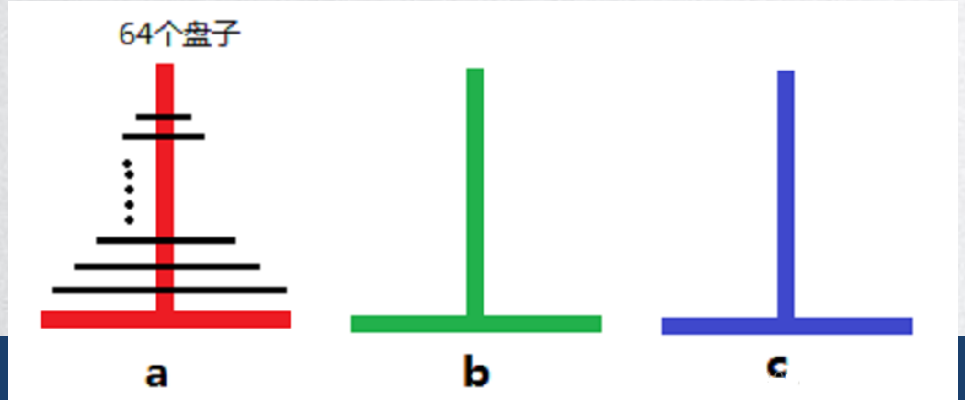
右边的展式是一个无穷级数，被称为**莱布尼茨级数**，这个级数收敛到$\frac{\pi}{4}$。

戈里。使用求和符号可记作：

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

课后练习2（请在未来课堂系统提交）

- 汉诺塔问题
  - 汉诺塔(Tower of Hanoi)源于印度传说中，大梵天创造世界时造了三根金钢石柱子，其中一根柱子自底向上叠着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。
  - 请用Python编写一个汉诺塔的移动函数，将所有圆盘从A移动到C，一次只能移动一个盘子，盘子只能在3个标杆之间移动，更大的盘子不能放在更小的盘子上面。要求输入汉诺塔的层数（有多少个盘子），输出整个移动流程。



谢谢！