A L G O R I T H M

P R A C T I C E

深度学习算法与实践

Steven Tang

渡一教育
Duyi Education

4

程 序 的 控 制 结 构

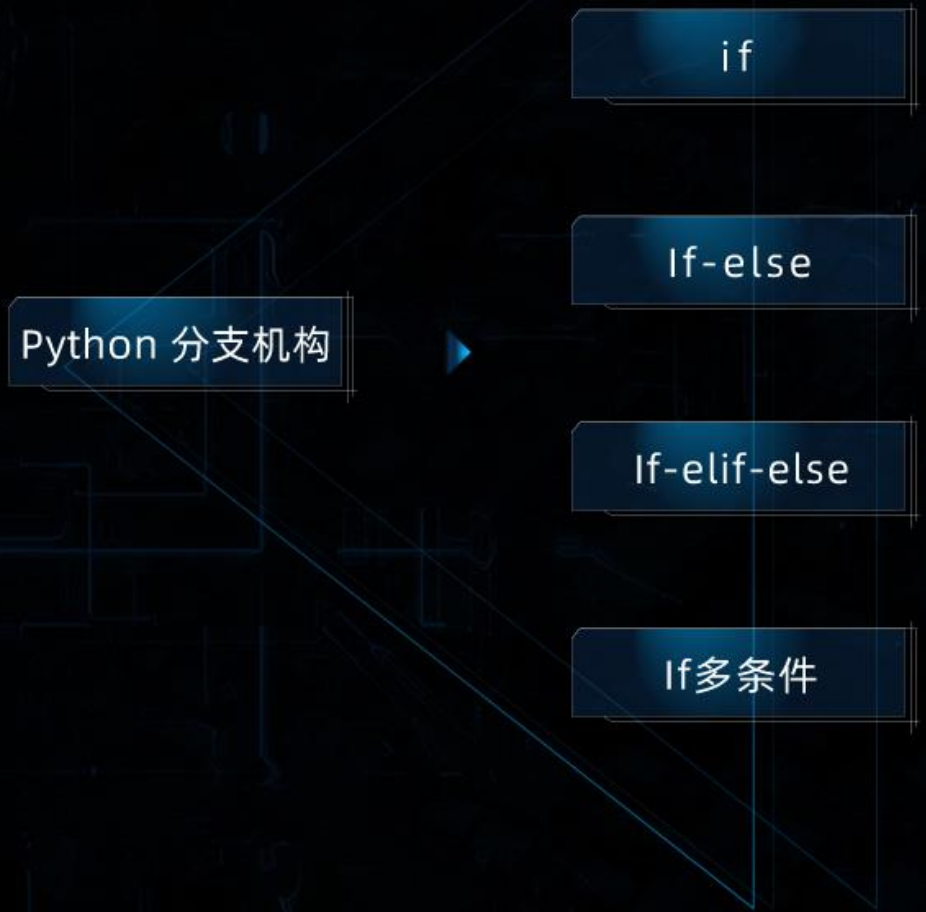
渡一教育
Duyi Education

程序的分支结构

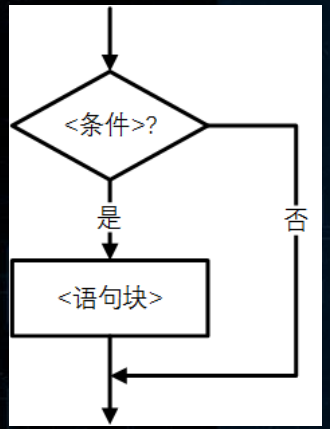
程序的循环结构

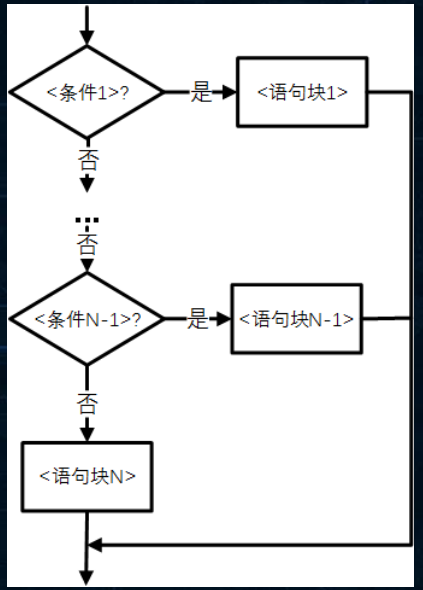
程序的异常处理

分支结构



单分支结构和多分支结构





分支结构的控制流程图

关系操作符

- 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

Python中浮点数的比较

```
In [11]: 0.2==0.2
Out[11]: True

In [12]: 0.3==0.1+0.2
Out[12]: False

In [13]: 0.1+0.2
Out[13]: 0.30000000000000004

In [14]: a=0.1+0.2

In [15]: abs(a-0.3)<0.00001
Out[15]: True

In [16]: import math

In [17]: math.isclose(a,0.3)
Out[17]: True
```

Python中多条件语句

```
In [23]: a=3

In [24]: a>2 and a<4
Out[24]: True

In [25]: 5>a>2
Out[25]: True

In [26]: 2<a<5
Out[26]: True

In [27]: 2<a>5
Out[27]: False
```

```
In [16]: a=True

In [17]: not a
Out[17]: False

In [18]: not not a
Out[18]: True
```

```
In [28]: b1=True

In [29]: b2=False

In [30]: b1 or b2
Out[30]: True

In [31]: b3=False

In [32]: b2 or b3
Out[32]: False

In [33]: b1 and b2
Out[33]: False

In [34]: b4=True

In [35]: b1 and b4
Out[35]: True
```

If-else语句实例

判断是否为闰年

```
import calendar


year = int(input("请输入年份："))
check_year=calendar.isleap(year)
if check_year == True:
    print ("闰年")
else:
    print ("非闰年")
```

分支结构单行表示

二分支结构还有一种更简洁的表达，适合通过判断返回特定值，语法格式如下：

<表达式1> if <条件> else <表达式2>

```
1  Temp = eval(input("请输入你的体温： "))
2  print("体温{}异常!".format("存在" if Temp >= 37.2 else "没有"))
```

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 = 3#类型可以不同
>>>count if count!=0 else "不存在"
3
>>>count = 0
>>>count if count!=0 else "不存在"
"不存在"
```

多分支结构: if-elif-else语句

```
1  BMI = eval(input("请输入你的BMI数值: "))
2  if 0<= BMI < 15:
3      print("你太瘦了，要多吃点!")
4  elif 15 <= BMI <25:
5      print("你的身材很好，继续保持哦！")
6  else:
7      print("你有点超重了，要减减肥！")
```

程序的分支结构

程序的循环结构

程序的异常处理

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>
```

遍历结构可以是字符串、组合数据类型或range()函数：

循环N次	遍历字符串s	遍历列表ls
for i in range(N): <语句块>	for c in s: <语句块>	for item in ls: <语句块>

range函数

range()是一个计数函数，实现循环从一个数字开始计数到另一个数字，一旦到达最后的数字或者某个条件不再满足就立刻退出循环。

range函数的语法如下：

range(start, end, step=1)

```
In [28]: a=range(10)

In [29]: type(a)
Out[29]: range

In [30]: list(a)
Out[30]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [31]: b=range(1,10,2)

In [32]: list(b)
Out[32]: [1, 3, 5, 7, 9]

In [33]:
```

enumerate枚举函数

枚举函数enumerate()用于将序列对象转换为一个索引序列，并返回序列对象的索引和成员，一般在for循环中得到序列对象的索引计数和序列对象成员。使用枚举函数迭代序列对象的语法如下：

for index, iter_var in enumerate (list,start_index=0):

```
In [33]: names=['张三','李四','王五']

In [34]: for index,name in enumerate(names):
    ...:     print("%d %s"%(index,name))
    ...:
0 张三
1 李四
2 王五
```

zip函数

zip函数接受任意多个（包括0个和1个）序列作为参数，返回一个tuple列表。

```
In [35]: names=['张三','李四','王五']

In [36]: ages=[15,16,22]

In [37]: for name,age in zip(names,ages):
    ...:     print("%s %d"%(name,age))
    ...:
张三 15
李四 16
王五 22
```

reversed函数

```
In [11]: list(reversed(range(10)))
Out[11]: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

In [12]: for i in reversed(range(10)):
    ...:     print(i)
    ...:
9
8
7
6
5
4
3
2
1
0
```

遍历循环: for语句

- 当for循环正常执行之后，程序会继续执行else语句中内容。else语句只在循环正常执行之后才执行并结束
- 因此，可以在<语句块2>中放置判断循环执行情况的语句。

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

```
>>>
循环进行中: S
循环进行中: K
循环进行中: Y
循环正常结束
```

while循环

while 循环：

while 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
```

while循环

- while循环也有一种使用保留字else的扩展模式：

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

```
1  s, index = "SKY", 0
2  while index < len(s):
3      print("循环进行中: " + s[index])
4      index += 1
5  else:
6      s = "这句输出时是不满足while条件后输出"
7  print(s)
```

```
>>>
循环进行中: S
循环进行中: K
循环进行中: Y
循环正常结束
```

循环保留字：break和continue

- 循环结构有两个辅助保留字：break和continue，它们用来辅助控制循环执行
- break用来跳出最内层for或while循环，脱离该循环后程序从循环后代吗继续续执行

```
for s in "SKY":
    for i in range(6):
        print(s, end="")
        if s=="K":
            break
```

```
>>>
SSSSSSKYYYYYY
```

其中，break语句跳出了最内层for循环，但仍然继续执行外层循环。每个break语句只有能力跳出当前层次循环

循环保留字：break和continue

- continue用来结束当前当次循环，即跳出循环体中下面尚未执行的语句，但不跳出当前循环。
- 对于while循环，继续求解循环条件。而对于for循环，程序流程接着遍历循环列表
- 对比continue　break语句，如下

```
1  for s in "PYTHON":
2      if s=="T":
3          continue
4      print(s, end="")
```

```
>>>
PYHON
```

```
1  for s in "PYTHON":
2      if s=="T":
3          break
4      print(s, end="")
```

```
>>>
PY
```

渡一教育 Duyi Education

循环保留字：break和continue

continue语句和break语句的区别是：

continue语句只结束本次循环，而不终止整个循环的执行。

break语句则是结束整个循环过程，不再判断执行循环的条件是否成立

```
1  for s in "PYTHON":
2      if s=="T":
3          continue
4      print(s, end="")
```

```
>>>
PYHON
```

```
1  for s in "PYTHON":
2      if s=="T":
3          break
4      print(s, end="")
```

```
>>>
PY
```

循环保留字：break和continue

- for循环和while循环中都存在一个else扩展用法。
- else中的语句块只在一种条件下执行，即for循环正常遍历了所有内容没有因为break或return而退出。
- continue保留字对else没有影响。看下面两个例子

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

```
>>>
PYHON正常退出
```

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

```
>>>
PY
```

random库的使用

random库概述

- random库python中用于生成随机数的函数库。
- 这个库提供了不同类型的随机数函数，所有函数都是基于最基本的random.random()函数扩展而来。

random库解析

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

random库解析

对random库的引用方法与math库一样，采用下面两种方式实现：

import random　或　from random import *

```
In [43]: from random import *

In [44]: random()
Out[44]: 0.3219778591175191

In [45]: uniform(1,10)
Out[45]: 1.9155196274114406

In [46]: uniform(1,20)
Out[46]: 12.113168602575906

In [47]: randrange(0,100,4)
Out[47]: 80

In [48]: choice(range(100))
Out[48]: 66
```

```
In [52]: ls=list(range(10))

In [53]: shuffle(ls)

In [54]: print(shuffle(ls))
None

In [55]: print(ls)
[9, 5, 2, 7, 8, 6, 3, 0, 4, 1]
```

基本随机数

Python中产生随机数使用随机数种子来产生（只要种子相同，产生的随机序列，无论是每一个数，还是数与数之间的关系都是确定的，所以随机数种子确定了随机序列的产生）

```
In [67]: seed(12)

In [68]: "{}-{}-{}".format(randint(1,10),randint(1,10),randint(1,10))
Out[68]: '8-5-9'

In [69]: "{}-{}-{}".format(randint(1,10),randint(1,10),randint(1,10))
Out[69]: '6-3-7'

In [70]: seed(12)

In [71]: "{}-{}-{}".format(randint(1,10),randint(1,10),randint(1,10))
Out[71]: '8-5-9'
```

π的计算

蒙特卡罗采样法

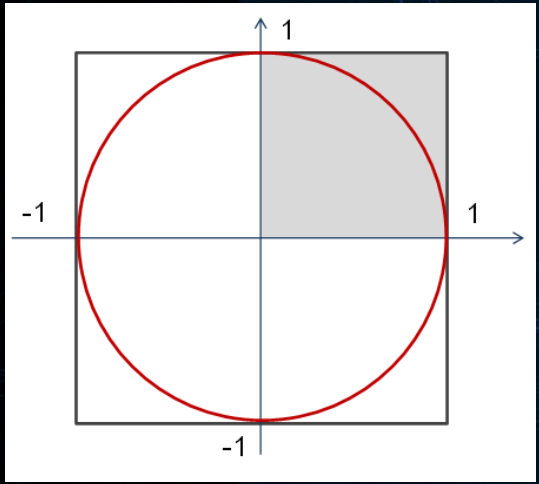
随着计算机的出现，数学家找到了另类求解π的另类方法：蒙特卡罗（Monte Carlo）方法，又称随机抽样或统计试验方法。

π的计算

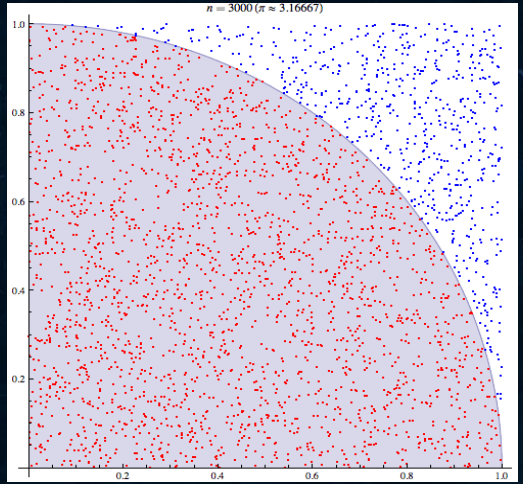
应用蒙特卡罗方法求解π的基本步骤如下：

- 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π/4值。

π的计算



计算π使用的正方形和圆结构



计算π使用的1/4区域和抛点过程

π的计算

```
1
2  import random
3  import time
4  DARTS = 1000000
5  hits = 0.0
6  start=time.time()
7  for i in range(1, DARTS+1):
8      x, y = random.random(), random.random()
9      dist = pow(x ** 2 + y ** 2,0.5)
10     if dist <= 1.0:
11         hits = hits + 1
12 pi = 4 * (hits/DARTS)
13 end=time.time()
14 print("Pi值是{}.".format(pi))
15 print("运行时间是:%.5f"%(end-start))
```

```
>>>
Pi值是3.140516.
运行时间是:0.81080
```

程序的分支结构

程序的循环结构

程序的异常处理

异常处理：try-except语句

观察下面这段小程序：

当用户输入的不是数字呢？

```
In [7]: num=eval(input("输一个数: "))

输一个数: 30

In [8]: num=eval(input("输一个数: "))

输一个数: 25.5

In [9]: num=eval(input("输一个数: "))

输一个数: NO
Traceback (most recent call last):

  File "<ipython-input-9-b30a343730e3>", line 1, in <module>
    num=eval(input("输一个数: "))

  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

```
输一个数：NO
Traceback (most recent call last):

  File "<ipython-input-12-b5dfe1d10a3c>", line 1, in <module>
    runfile('C:/Users/Dr. Tang/Desktop/渡一/Code lesson3/exception.py', wdir='C:/Users/Dr. Tang/Desktop/渡一/Code lesson3')

  File "D:\Anaconda3\envs\duyi\lib\site-packages\spyder_kernels\customize\spydercustomize.py", line 827, in runfile
    execfile(filename, namespace)

  File "D:\Anaconda3\envs\duyi\lib\site-packages\spyder_kernels\customize\spydercustomize.py", line 110, in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "C:/Users/Dr. Tang/Desktop/渡一/Code lesson3/exception.py", line 3, in <module>
    num=eval(input("输一个数："))

  File "<string>", line 1, in <module>

NameError: name 'NO' is not defined
```

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

```
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!")
```

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

```
try:
    letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    idx = eval(input("请输入一个整数: "))
    print(letters[idx])
except NameError:
    print("输入错误，请输入一个整数!")
except:
    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01到256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：

```
try:
    letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    idx = eval(input("请输入一个整数: "))
    print(letters[idx])
except NameError:# 异常时，执行该块
    print("输入错误，请输入一个整数!")
except IndexError:# 异常时，执行该块
    print("输入错误，请输入0-25中的一个整数!")
else:# 主代码块执行完，执行该块
    print("没有发生异常")
finally:#无论异常与否，最终执行该块
    print("程序执行完毕")
```

异常的高级用法

先特殊后万能

```
try:
    letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    idx = eval(input("请输入一个整数: "))
    print(letters[idx])
except NameError:# 异常时，执行该块
    print("输入错误，请输入一个整数!")
except IndexError:# 异常时，执行该块
    print("输入错误，请输入0-25中的一个整数!")
except Exception:# 异常时，执行该块
    print("其他错误！请重新输入！")
else:# 主代码块执行完，执行该块
    print("没有发生异常")
finally:#无论异常与否，最终执行该块
    print("程序执行完毕")
```

异常的高级用法

```
s1 = 'hello'
try:
    int(s1)
except KeyError as ke:
    print ('键错误:',ke)
except IndexError as ie:
    print ('索引错误:',ie)
except Exception as e:
    print ('其他错误:',e)
```

异常的高级用法

```
try:
    s = None
    if s is None:
        print ("s 是空对象")
        raise Exception     #如果
    print (len(s))  #这句不会执行,
except TypeError as te:
    print("类型错误: ",te)
    print ("空对象没有长度")

s = None
if s is None:
    raise NameError
print ('正常结束')
```

```
BaseException # 所有异常的基类
 +-- SystemExit # 解释器请求退出
 +-- KeyboardInterrupt # 用户中断执行(通常是输入^C)
 +-- GeneratorExit # 生成器(generator)发生异常来通知退出
 +-- Exception # 常规异常的基类
      +-- StopIteration # 迭代器没有更多的值
      +-- StopAsyncIteration # 必须通过异步迭代器对象的__anext__()方法引发以停止迭代
      +-- ArithmeticError # 各种算术错误引发的内置异常的基类
      |    +-- FloatingPointError # 浮点计算错误
      |    +-- OverflowError # 数值运算结果太大无法表示
      |    +-- ZeroDivisionError # 除(或取模)零 (所有数据类型)
      +-- AssertionError # 当assert语句失败时引发
      +-- AttributeError # 属性引用或赋值失败
      +-- BufferError # 无法执行与缓冲区相关的操作时引发
      +-- EOFError # 当input()函数在没有读取任何数据的情况下达到文件结束条件(EOF)时引发
      +-- ImportError # 导入模块/对象失败
      |    +-- ModuleNotFoundError # 无法找到模块或在在sys.modules中找到None
      +-- LookupError # 映射或序列上使用的键或索引无效时引发的异常的基类
      |    +-- IndexError # 序列中没有此索引(index)
      |    +-- KeyError # 映射中没有这个键
```

+-- MemoryError # 内存溢出错误(对于Python 解释器不是致命的)
+-- NameError # 未声明/初始化对象 (没有属性)
| +-- UnboundLocalError # 访问未初始化的本地变量
+-- OSError # 操作系统错误，EnvironmentError，IOError，WindowsError，socket.error，select.error和mmap.error已合并到OSError中，构造函数可能返回子类
| +-- BlockingIOError # 操作将阻塞对象(e.g. socket)设置为非阻塞操作
| +-- ChildProcessError # 在子进程上的操作失败
| +-- ConnectionError # 与连接相关的异常的基类
| | +-- BrokenPipeError # 另一端关闭时尝试写入管道或试图在已关闭写入的套接字上写入
| | +-- ConnectionAbortedError # 连接尝试被对等方中止
| | +-- ConnectionRefusedError # 连接尝试被对等方拒绝
| | +-- ConnectionResetError # 连接由对等方重置
| +-- FileExistsError # 创建已存在的文件或目录
| +-- FileNotFoundError # 请求不存在的文件或目录
| +-- InterruptedError # 系统调用被输入信号中断
| +-- IsADirectoryError # 在目录上请求文件操作(例如 os.remove())
| +-- NotADirectoryError # 在不是目录的事物上请求目录操作(例如 os.listdir())
| +-- PermissionError # 尝试在没有足够访问权限的情况下运行操作
| +-- ProcessLookupError # 给定进程不存在
| +-- TimeoutError # 系统函数在系统级别超时
+-- ReferenceError # weakref.proxy()函数创建的弱引用试图访问已经垃圾回收了的对象
+-- RuntimeError # 在检测到不属于任何其他类别的错误时触发
| +-- NotImplementedError # 在用户定义的基类中，抽象方法要求派生类重写该方法或者正在开发的类指示仍然需要添加实际实现

```
|  +-- RecursionError # 解释器检测到超出最大递归深度
   +-- SyntaxError # Python 语法错误
   |  +-- IndentationError # 缩进错误
   |       +-- TabError # Tab和空格混用
   +-- SystemError # 解释器发现内部错误
   +-- TypeError # 操作或函数应用于不适当类型的对象
   +-- ValueError # 操作或函数接收到具有正确类型但值不合适的参数
   |  +-- UnicodeError # 发生与Unicode相关的编码或解码错误
   |       +-- UnicodeDecodeError # Unicode解码错误
   |       +-- UnicodeEncodeError # Unicode编码错误
   |       +-- UnicodeTranslateError # Unicode转码错误
   +-- Warning # 警告的基类
        +-- DeprecationWarning # 有关已弃用功能的警告的基类
        +-- PendingDeprecationWarning # 有关不推荐使用功能的警告的基类
        +-- RuntimeWarning # 有关可疑的运行时行为的警告的基类
        +-- SyntaxWarning # 关于可疑语法警告的基类
        +-- UserWarning # 用户代码生成警告的基类
        +-- FutureWarning # 有关已弃用功能的警告的基类
        +-- ImportWarning # 关于模块导入时可能出错的警告的基类
        +-- UnicodeWarning # 与Unicode相关的警告的基类
        +-- BytesWarning # 与bytes和bytearray相关的警告的基类
        +-- ResourceWarning # 与资源使用相关的警告的基类。被默认警告过滤器忽略。
```

课后测试题

1. 猜年龄游戏

要求：

允许用户最多尝试3次，3次都没猜对的话，就直接退出，如果猜对了，打印 **恭喜** 并退出，如果没猜对，打印 **很遗憾！**

2. 密码输入程序

要求：

使用**try….except….else…finally**编写一个用户密码输入程序，要求全部为数字，要求长度为8位，不符合要求时必须抛出异常，可以用**raise**方法。