Python实验实例

原创

浩铖 于 2019-06-27 19:28:05 发布 13137 收藏 96
分类专栏： Python

Python 运算符、内置函数

实验目的：

1、熟练运用 Python 运算符。
2、熟练运用 Python 内置函数。

实验内容：

1、编写程序，输入任意大的自然数，输出各位数字之和。
2、编写程序，输入两个集合 setA 和 setB，分别输出它们的交集、并集和差集 setA-setB。
3、编写程序，输入一个自然数，输出它的二进制、八进制、十六进制表示形式。

```
num = input("请输入一个自然数：")
print(sum(map(int, num)))
# sum() 表示求和
# map(int,num) 表示将num的各位转换成整数
```

```
请输入一个自然数：13
4
```

```
setA = eval(input("请输入一个集合："))
setB = eval(input("请输入一个集合："))
print("交集：", setA | setB)
print("并集：", setA & setB)
print("setA-setB：", setA - setB)
```

```
请输入一个集合：{1,2,3,4}
请输入一个集合：{2,3,4,5}
交集： {1, 2, 3, 4, 5}
并集： {2, 3, 4}
setA-setB： {1}
```

```
num = int(input("请输入一个自然数："))
print("二进制：",bin(num))
print("十进制：",oct(num))
print("十六进制：",hex(num))
```

```
请输入一个自然数：12
二进制：  0b1100
十进制：  0o14
十六进制：  0xc
```

使用蒙特·卡罗方法计算圆周率近似值

实验目的：

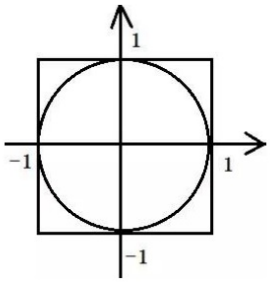
1、理解蒙特·卡罗方法原理。

2、理解 for 循环本质与工作原理。

3、了解 random 模块中常用函数。

实验内容：

蒙特·卡罗方法是一种通过概率来得到问题近似解的方法，在很多领域都有重要的应用，其中就包括圆周率近似值的计算问题。假设有一块边长为 2 的正方形木板，上面画一个单位圆，然后随意往木板上扔飞镖，落点坐标(x, y)必然在木板上（更多的时候是落在单位圆内），如果扔的次数足够多，那么落在单位圆内的次数除以总次数再乘以 4，这个数字会无限逼近圆周率的值。这就是蒙特·卡罗发明的用于计算圆周率近似值的方法，如图所示。



编写程序，模拟蒙特·卡罗计算圆周率近似值的方法，输入掷飞镖次数，然后输出圆周率近似值。

```
# author : "王佳伟"
# date : 2019-06-26
# File    : 3.py
# IDE：PyCharm

from random import random

times = int(input("请输入您投掷飞镖的次数："))
hit = 0
for i in range(times):
    x = random()
    y = random()
    if x*x+y*y<=1:
        hit=hit+1
print(4.0*hit/times)

#如果 x*x+y*y<=1 则说明生成的随机数（投掷的飞镖）落在了圆中。
#即投掷飞镖的点距离圆心的距离小于1
```

```
请输入您投掷飞镖的次数：100
3.16
```

小明爬楼梯

实验目的：

1、理解并熟练使用序列解包。

2、理解递归函数工作原理。

3、能够编写递归函数代码解决实际问题。

4、理解 Python 字典的用法。

实验内容：

假设一段楼梯共 15 个台阶，小明一步最多能上 3 个台阶。编写程序计算小明上这段楼梯一共有多少种方法。要求给出递推法和递归法两种代码。

```
# author : "王佳伟"
# date : 2019-06-26
# File    : 7.py
# IDE：PyCharm

'''
递推法：
从第15个台阶上往回看，有3种方法可以上来（从第14个台阶上一步迈1个台阶上来，
从第13个台阶上一步迈2个台阶上来，从第12个台阶上一步迈3个台阶上来），
同理，第14个、13个、12个台阶都可以这样推算，从而得到公式f(n) = f(n-1) + f(n-2) + f(n-3)，
其中n=15、14、13、...、5、4。
然后就是确定这个递归公式的结束条件了，
第一个台阶只有1种上法，第二个台阶有2种上法（一步迈2个台阶上去、一步迈1个台阶分两步上去），
第三个台阶有4种上法。
'''
def climbStairs1(n):
    # 递推法
    a = 1  # 上一个台阶只有一种方法
    b = 2  # 上两个台阶有两种方法
    c = 4  # 上三个台阶有四种方法
    for i in range(n - 3):
        c, b, a = a + b + c, c, b
    return c

def climbStairs2(n):
    # 递归法
    first3 = {1:1,2:2,3:4}
    if n in first3.keys():
        return first3[n]
    else:
        return climbStairs2(n-1)+climbStairs2(n-2)+climbStairs2(n-3)

print(climbStairs1(15))
print(climbStairs2(15))
```

```
5768
5768
```

蒙蒂霍尔悖论游戏

实验目的：

1、了解蒙蒂霍尔悖论内容。

2、了解游戏规则。

3、熟练运用字典方法和集合运算。

4、了解断言语句 assert 的用法。

5、熟练运用循环结构。

实验内容：

假设你正参加一个有奖游戏节目，并且有 3 道门可选：其中一个后面是汽车，另外两个后面是山羊。你选择一个门，比如说 1 号门，主持人当然知道每个门后面是什么并且打开了另一个门，比如说 3 号门，后面是一只山羊。这时，主持人会问你"你想改选 2 号门吗？"，然后根据你的选择确定最终要打开的门，并确定你获得山羊（输）或者汽车（赢）。

编写程序，模拟上面的游戏。

```
# author : "王佳伟"
# date : 2019-06-26
# File    : 9.py
# IDE：PyCharm

from random import randrange

def init():
    '''
    初始化三个门以及后面门的物品
    :return: 返回三个门后对应的物品
    '''
    result = {i: 'goat' for i in range(3)}  # {0:goat,1:goat,2:goat}
    r = randrange(3) # 随机生成0~2之间的一个整数
    result[r] = 'car' # 将随机索引位置的goat改为car
    return result # 初始化门完成，返回门的集合

def StartGame():
    '''
    开始游戏
    :return: 返回游戏结果
    '''
    doors = init() # 初始化门
    # 获取玩家选择的门号
    while True:
        try:
            firstDoorNum = int(input("Choose a door to open:"))
            # assert —— Python的断言就是检测一个条件，如果条件为真，它什么都不做；反之它触发一个带可选错误信息的Asser
            assert 0<= firstDoorNum <=2
            break
        except:
            print("Door number must be between {} and {} :".format(0,2))
    # 主持人查看另外两个门后边的物品情况
    # 从所有的门（三个门—— 0,1,2）的索引中删除玩家选中的们的所引
    for door in doors.keys() - {firstDoorNum}:
        # 打开其中一个门后为羊的门
        if doors[door] == 'goat':
            print("goat behind the door",door)
            # 获取第三个门号，让玩家纠结
            # S.pop() 随机返回集合S中的一个元素，如果S为空，产生异常
            thirDoor = (doors.keys()-{door,firstDoorNum}).pop()
            change = input("Switch to {} y/n: ".format(thirDoor))
            # 如果change='y'则finalDoorNum = thirDoor，否则finalDoorNum = firstDoorNum
            finalDoorNum = thirDoor if change=='y' else firstDoorNum
            if doors[finalDoorNum] == 'goat':
                return "I Win !"
            else:
                return "You Win !"

while True:
    print("="*30)
    print(StartGame())
    r = input("Do you want to try once more (y/n) :")
    if r == n:
        break
```

```
==============================
Choose a door to open:1
goat behind the door 0
Switch to 2 y/n: n
I Win !
Do you want to try once more (y/n) :
```

猜数游戏

实验目的：

1、熟练运用选择结构与循环结构解决实际问题。
2、注意选择结构嵌套时代码的缩进与对齐。
3、理解带 else 子句的循环结构执行流程。
4、理解条件表达式 value1 if condition else value2 的用法。
5、理解使用异常处理结构约束用户输入的用法。
6、理解带 else 子句的异常处理结构的执行流程。

实验内容：

编写程序模拟猜数游戏。程序运行时，系统生成一个随机数，然后提示用户进行猜测并根据用户输入进行必要的提示（猜对了、太大了、太小了），如果猜对则提前结束程序，如果次数用完仍没有猜对，提示游戏结束并给出正确答案。

```
# author : "王佳伟"
# date : 2019-06-26
# File    : 10.py
# IDE：PyCharm

from random import randint

def guessNumber(maxValue=10,maxTimes=3):
    '''
    猜数字
    :param maxValue:随机生成最大的数字
    :param maxTimes:猜数字的次数
    :return:null
    '''
    # 随机生成一个整数 —— 1到 maxValue
    value = randint(1,maxValue)
    for i in range(maxTimes):
        # 如果i=1则输出Start to GUESS：提示，否则输出GUESS again：提示。
        prompt = 'Start to GUESS：' if i==0 else 'GUESS again：'
        # 使用异常处理结构，防止输入的不是数字。
        try:
            x = int(input(prompt)) # 获取到玩家输入的值
        except:
            print("Must input an interger between 1 and",maxValue)
        else:
            if x == value:
                # 猜对了
                print("Congratulations! ")
                break
            elif x>value:
                print("Too big")
            else:
                print("Too litter")
    else:
        # else 与 for连用，表示for循环结束后运行else
        # 次数用完还没有猜对，游戏结束，给出提示。
        print("Game over , FAIL.")
        print("The number is ", value)

guessNumber()
```

```
Start to GUESS：5
Too litter
GUESS again：7
Too litter
GUESS again：9
Too litter
Game over , FAIL.
The number is  10
```

抓狐狸游戏

实验目的：

1、培养分析问题并对进行建模的能力。

2、熟练使用列表解决实际问题。

3、熟练运用选择结构和循环结构解决实际问题。

4、理解带 else 子句的循环结构执行流程。

5、理解使用异常处理结构约束用户输入的用法。

实验内容：

编写程序，模拟抓狐狸小游戏。假设一共有一排 5 个洞口，小狐狸最开始的时候在其中一个洞口，然后玩家随机打开一个洞口，如果里面有狐狸就抓到了。如果洞口里没有狐狸就第二天再来抓，但是第二天狐狸会在玩家来抓之前跳到隔壁洞口里。

```
# author : "王佳伟"
# date : 2019-06-26
# File    : 11.py
# IDE：PyCharm

from random import choice, randrange


def catchMe(n=5, maxStep=10):
    '''
    抓狐狸游戏
    :param n: 洞口的个数，默认为5
    :param maxStep: 最大抓取次数，默认允许抓10次
    :return: null
    '''
    # 共有n个洞口，设有狐狸的为1，没有的为0
    positions = [0] * n
    # 狐狸的随机初始位置
    oldPos = randrange(0, n)
    positions[oldPos] = 1

    # 抓maxStep次
    while maxStep >= 0:
        maxStep -= 1
        # 这个循环是为了保证用户输入是否是有效的洞口
        while True:
            try:
                x = input("今天打算打开哪个洞口？（0-{0}）:".format(n - 1))
                # 如果输入的不是数字，会跳转到except部分
                x = int(x)
                # 如果输入的洞口有效，结束这个循环，否则继续输入
                assert 0 <= x < n, "要按套路来啊，再给你一次机会哈！"
                break
            except:
                # 如果输入的不是数字，就执行这里的代码
                print("你要按套路来啊！再给你一次机会！")
        if positions[x] == 1:
            print("成功！我抓到狐狸了！")
            break
        else:
            print("今天没有抓到狐狸。")

        # 如果这次没抓到，则狐狸就会跳到隔壁的洞口
        # 分三种情况：如果在最左边，则只能往右边洞口跳
        # 如果在最右边，则只能往左边洞口跳
        # 如果在中间三个洞头则随机向左右两边洞口跳
```

```
        if oldPos == n-1: # 在最右边
            newPos = oldPos -1
        elif oldPos ==0: # 在最左边
            newPos = oldPos + 1
        else:
            newPos = oldPos + choice((-1,1))
        positions[oldPos],positions[newPos] = 0,1
        oldPos = newPos
    else:
        print("十天了！放弃吧！你这样瞎鸡巴抓是没有希望的！")

# 启动游戏
catchMe()
```

```
今天打算打开哪个洞口？（0-4）：3
今天又没抓到。
今天打算打开哪个洞口？（0-4）：1
今天又没抓到。
今天打算打开哪个洞口？（0-4）：6
要按套路来啊，再给你一次机会。
今天打算打开哪个洞口？（0-4）：5
要按套路来啊，再给你一次机会。
今天打算打开哪个洞口？（0-4）：4
今天又没抓到。
今天打算打开哪个洞口？（0-4）：2
今天又没抓到。
今天打算打开哪个洞口？（0-4）：0
今天又没抓到。
今天打算打开哪个洞口？（0-4）：3
今天又没抓到。
今天打算打开哪个洞口？（0-4）：2
今天又没抓到。
今天打算打开哪个洞口？（0-4）：1
今天又没抓到。
今天打算打开哪个洞口？（0-4）：1
成功，我抓到小狐狸啦。
```

凯撒加密

实验目的：

1、了解 Python 标准库 string。

2、理解凯撒加密算法原理。

3、理解切片操作。

4、熟练运用字符串对象的方法。

实验内容：

编写程序，要求输入一个字符串，然后输入一个整数作为凯撒加密算法的密钥，然后输出该字符串加密后的结果。

```
# author : "王佳伟"
# date : 2019-06-26
# File    : 1.py
# IDE：PyCharm

import string


def kaisa(s, k):
    lower = string.ascii_lowercase
    upper = string.ascii_uppercase
    before = string.ascii_letters
    after = lower[k:] + lower[:k] + upper[k:] + upper[:k]
    table = ''.maketrans(before, after)
    return s.translate(table)


s = input("请输入一个字符串：")
k = int(input("请输入一个整数密钥："))
print(kaisa(s, k))

#ascii_lowercase = 'abcdefghijklmnopqrstuvwxyz'
#ascii_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
#ascii_letters = ascii_lowercase + ascii_uppercase
```

```
请输入一个字符串：dsfgh
请输入一个整数密钥：2
fuhij
```

使用列表实现筛选法求素数

实验目的：

1、理解筛选法求解素数的原理。
2、理解列表切片操作。
3、熟练运用内置函数 enumerate()。
4、熟练运用内置函数 filer()。
5、理解序列解包工作原理。
6、初步了解选择结构和循环结构。

实验内容：

编写程序，输入一个大于 2 的自然数，然后输出小于该数字的所有素数组成的列表。

```
# author : "王佳伟"
# date : 2019-06-28
# File    : 4.py
# IDE：PyCharm

maxNumber = int(input("请输入一个大于2的自然数："))
lst = list(range(2,maxNumber))
# 最大的整数平方根
m = int(maxNumber**0.5)
for index,value in enumerate(lst):
    # 如果当前数字已经大于做大整数平方根，结束判断
    if value > m:
        break
    #对改位置之后的元素进行过滤
    lst[index+1:] = filter(lambda x:x%value!=0,lst[index+1:])
print(lst)

# enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 f
# filter() 函数用于过滤序列，过滤掉不符合条件的元素，返回由符合条件元素组成的新列表。
```

```
请输入一个大于2的自然数：16
[2, 3, 5, 7, 11, 13]
```

汉诺塔问题

实验目的：

1、理解函数默认值参数。

2、理解函数递归。

3、熟练运行列表对象的方法。

实验内容：

据说古代有一个梵塔，塔内有三个底座 A、B、C，A 座上有 64 个盘子，盘子大小不等，大的在下，小的在上。有一个和尚想把这 64 个盘子从 A 座移到 C 座，但每次只能允许移动一个盘子。在移动盘子的过程中可以利用 B 座，但任何时刻 3 个座上的盘子都必须始终保持大盘在下、小盘在上的顺序。如果只有一个盘子，则不需要利用 B 座，直接将盘子从 A 移动到 C 即可。编写函数，接收一个表示盘子数量的参数和分别表示源、目标、临时底座的参数，然后输出详细移动步骤和每次移动后三个底座上的盘子分布情况。

```
# author : "王佳伟"
# date : 2019-06-28
# File    : 12.py
# IDE：PyCharm

def haonoi(num, src, dst, temp=None):  # 递归算法
    if num < 1:
        return
    global times  # 声明用来记录移动次数的变量为全局变量
    # 递归调用函数自身，先把除最后一个盘子以外的所有盘子移动到临时柱子上
    haonoi(num - 1, src, temp, dst)
    # 移动最有一个盘子
    print("the {0} times move:{1} - -> {2}".format(times, src, dst))
    towers[dst].append(towers[src].pop())
    for tower in "ABC":  # 输出三根柱子上的盘子
        print(tower, ":", towers[tower])
    times += 1
    # 把除最后一个盘子之外的其他盘子从临时柱子上移动到目标柱子上
    haonoi(num - 1, temp, dst, src)


times = 1  # 用来记录移动次数
n = 3  # 盘子数量
towers = {'A': list(range(n, 0, -1)),  # 初始状态，所有盘子都在A柱上
          'B': [],
          'C': []
          }
# A表示最初防止盘子的柱子，C表示目标柱子，B是临时柱子
haonoi(n, 'A', 'C', 'B')
```

```
the 1 times move:A - -> C
A : [3, 2]
B : []
C : [1]
the 2 times move:A - -> B
A : [3]
B : [2]
C : [1]
the 3 times move:C - -> B
A : [3]
B : [2, 1]
C : []
the 4 times move:A - -> C
A : []
B : [2, 1]
C : [3]
the 5 times move:B - -> A
A : [1]
B : [2]
C : [3]
the 6 times move:B - -> C
A : [1]
B : []
C : [3, 2]
the 7 times move:A - -> C
A : []
B : []
C : [3, 2, 1]
```