Python Tutorial 1

```
print('Hello World')
radius = 5
area = 3.14*radius**2
print ('A circle of radius',radius,'has an area of',area)
```

```
>>> %Run -c $EDITOR_CONTENT
Hello World
A circle of radius 5 has an area of 78.5
```

Python Tutorial 2

```
picture = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
print (picture[1])
print (picture[1][2])
x = []
x.append(50)
print(x)
x.append(60)
print(x)
a = 1
b = 2
c = 3
d = 4
e = 5
grads=[a,b,c,d,e]
print(grads)
```

```
>>> %Run -c $EDITOR_CONTENT
[5, 6, 7, 8]
7
[50]
[50, 60]
[1, 2, 3, 4, 5]
```

Python Tutorial 3

```
x = 7
y = 2
z = x+y
print(x,'+',y,'=',z)
num = input('Please Input Your Number: ')
print('Your Number is ',num)
name = input('Please Enter Your Name: ')
print('Hello',name,', Welcome to Python')
```

```
>>> %Run -c $EDITOR_CONTENT
7 + 2 = 9
Please Input Your Number: 6
Your Number is  6
Please Enter Your Name: Chunhei
Hello Chunhei , Welcome to Python
```

Homework

```
num1 = input('Please Input Your First Number')
num2 = input('Please Input Your Second Number')
an = num1 + num2
print (num1,' + ',num2,' = ',an)
```

```
>>> %Run -c $EDITOR_CONTENT
Please Input Your First Number5
Please Input Your Second Number6
5  +  6  =  56
```

Python Tutorial 4

```
num1 = float(input('Please Enter Your First Number'))
num2 = float(input('Please Enter Your Second Number'))
an = num1 + num2
print (num1,' + ',num2,' = ',an)
```

```
>>> %Run -c $EDITOR_CONTENT
```

```
Please Enter Your First Number5.5
Please Enter Your Second Number6.6
5.5  +  6.6  =  12.1
```

Python Tutorial 5

Install Visual Studo Code.

Python Tutorial 6

```
mynumber = float(input('Please Input Your Number: '))
rem = mynumber%2
if (rem==0):
    print('You have and Even Number')
    print('Please Play Again!')
if (rem==1):
    print('You have and Odd Number')
    print('Please Play Again!')
```

```
>>> %Run -c $EDITOR_CONTENT
Please Input Your Number: 5
You have and Odd Number
Please Play Again!
```

```
>>> %Run -c $EDITOR_CONTENT
Please Input Your Number: 6
You have and Even Number
Please Play Again!
```

Python Tutorial 7

```
mynum = float(input('Please Input Your Number'))
if (mynum>=5 and mynum<=10):
    print('Congratulations, Your Number is Between 5 and 10')
```

```
if (mynum<5 or mynum>10):
    print('Sorry,Your Number is Not between 5 and 10')
```

```
>>> %Run -c $EDITOR_CONTENT
Please Input Your Number6
Congratulations, Your Number is Between 5 and 10

%Run -c $EDITOR_CONTENT
Please Input Your Number11
Sorry,Your Number is Not between 5 and 10

>>> %Run -c $EDITOR_CONTENT
Please Input Your Number1
Sorry,Your Number is Not between 5 and 10
```

Python Tutorial 8

```
mynum = float(input('Please Input Your Number: '))
rem = mynum%2
if (mynum>0 and rem==0):
    print ('You Have an Even Positive Number')
if (mynum>0 and rem== 1):
    print ('You Have an odd Positive Number')
if (mynum<0 and rem==0):
    print ('You Have an Even Negative Number')
if (mynum<0 and rem==1):
    print ('You Have an Odd Negative Number')
if (mynum==0):
    print('Your Number is Zero')
```

```
>>> %Run -c $EDITOR_CONTENT
Please Input Your Number: 5
You Have an odd Positive Number

>>> %Run -c $EDITOR_CONTENT
Please Input Your Number: -6
You Have an Even Negative Number

>>> %Run -c $EDITOR_CONTENT
Please Input Your Number: 8
You Have an Even Positive Number
```

```
>>> %Run -c $EDITOR_CONTENT
Please Input Your Number: -11
You Have an Odd Negative Number

>>> %Run -c $EDITOR_CONTENT
Please Input Your Number: 0
Your Number is Zero
```

Python Tutorial 9

```
numgrades = int (input('How Many Geades Do Ypu Have? '))
grades = []
for i in range(0,numgrades,1):
   grade = float(input('Please Enter Your Grade: '))
   grades.append(grade)
   print(grades)
print('Your Grades Are: ')
for i in range(0,numgrades,1):
   print(grades[i])
print('That All folks')
```

```
>>EDITOR_CONTENT> %Run -c $
How Many Geades Do Ypu Have? 3
Please Enter Your Grade: 20
[20.0]
Please Enter Your Grade: 30
[20.0, 30.0]
Please Enter Your Grade: 50
[20.0, 30.0, 50.0]
Your Grades Are:
20.0
30.0
50.0
That All folks
```

Homework

```
numgrades = int (input('How Many Grades Do Ypu Have? '))
grades = []
Grades = 0.0
```

```
for i in range(0,numgrades,1):
    grade = float(input('Please Enter Your Grade: '))
    grades.append(grade)
    print(grades)
print('Your Grades Are: ')
for i in range(0,numgrades,1):
    print(grades[i])
for a in range(0,numgrades,1):
    Grades = Grades + grades[a]
Average = (Grades/numgrades)
print ('Your Average is: ',Average)
```

```
%Run -c $EDITOR_CONTENT
How Many Grades Do Ypu Have? 3
Please Enter Your Grade: 97
[97.0]
Please Enter Your Grade: 89
[97.0, 89.0]
Please Enter Your Grade: 93
[97.0, 89.0, 93.0]
Your Grades Are:
97.0
89.0
93.0
Your Average is:  93.0
```

Python Tutorial 10

```
numgrades = int (input('How Many Grades Do Ypu Have? '))
grades = []
Grades = 0.0
for i in range(0,numgrades,1):
    grade = float(input('Please Enter Your Grade: '))
    grades.append(grade)
    print(grades)
print('Your Grades Are: ')
for i in range(0,numgrades,1):
    print(grades[i])
for a in range(0,numgrades,1):
    Grades = Grades + grades[a]
Average = Grades/numgrades
```

```
print('')
print ('Your Average is: ',Average)
```

```
>>> %Run -c $EDITOR_CONTENT
How Many Grades Do Ypu Have? 3
Please Enter Your Grade: 97
[97.0]
Please Enter Your Grade: 98
[97.0, 98.0]
Please Enter Your Grade: 99
[97.0, 98.0, 99.0]
Your Grades Are:
97.0
98.0
99.0

Your Average is:  98.0
```

Python Tutorial 11

```
numgrades = int (input('How Many Grades Do Ypu Have? '))
grades = []
Grades = 0.0
lowGrade = 100
highGrade = 0
for i in range(0,numgrades,1):
    grade = float(input('Please Enter Your Grade: '))
    grades.append(grade)
    print(grades)
print('Your Grades Are: ')
for i in range(0,numgrades,1):
    print(grades[i])
for a in range(0,numgrades,1):
    Grades = Grades + grades[a]
Average = Grades/numgrades
print('')
print('Your Average is: ',Average)
for i in range(0,numgrades,1):
    if grades[i]<lowGrade:
        lowGrade = grades[i]
print('Your Low Grade is: ',lowGrade)
for  i in range(0,numgrades,1):
    if grades[i]>highGrade:
        highGrade = grades[i]
print('Your High Grade is: ',highGrade)
```

```
>>> %Run -c $EDITOR_CONTENT
How Many Grades Do Ypu Have? 5
Please Enter Your Grade: 85
[85.0]
Please Enter Your Grade: 90
[85.0, 90.0]
Please Enter Your Grade: 95
[85.0, 90.0, 95.0]
Please Enter Your Grade: 100
[85.0, 90.0, 95.0, 100.0]
Please Enter Your Grade: 80
[85.0, 90.0, 95.0, 100.0, 80.0]
Your Grades Are:
85.0
90.0
95.0
100.0
80.0

Your Average is:  90.0
Your Low Grade is:  80.0
Your High Grade is:  100.0
```

Python Tutorial 12

```
numgrades = int（ input('How Many Grades Do You Have? '))
grades = []
Grades = 0.0
lowGrade = 100
highGrade = 0
for i in range(0,numgrades,1):
    grade = float(input('Please Enter Your Grade: '))
    grades.append(grade)
    print(grades)
print('Your Grades Are: ')
for i in range(0,numgrades,1):
    print(grades[i])
```

```
for a in range(0,numgrades,1):
    Grades = Grades + grades[a]
Average = Grades/numgrades
print('')
print('Your Average is: ',Average)
for i in range(0,numgrades,1):
    if grades[i]<lowGrade:
        lowGrade = grades[i]
print('Your Low Grade is: ',lowGrade)
for  i in range(0,numgrades,1):
    if grades[i]>highGrade:
        highGrade = grades[i]
print('Your High Grade is: ',highGrade)
for i in range(0,numgrades-1,1):
    for i in range(0,numgrades-1,1):
       if grades[i]<grades[i+1]:
           swp=grades[i]
           grades[i]=grades[i+1]
           grades[i+1]=swp
print ('Your Sorted Grade List is: ')
for i in range(0,numgrades,1):
    print(grades[i])
```

```
>>> %Run -c $EDITOR_CONTENT
How Many Grades Do You Have? 5
Please Enter Your Grade: 97
[97.0]
Please Enter Your Grade: 95
[97.0, 95.0]
Please Enter Your Grade: 99
[97.0, 95.0, 99.0]
Please Enter Your Grade: 93
[97.0, 95.0, 99.0, 93.0]
Please Enter Your Grade: 100
[97.0, 95.0, 99.0, 93.0, 100.0]
Your Grades Are:
97.0
95.0
99.0
93.0
100.0

Your Average is:  96.8
Your Low Grade is:  93.0
Your High Grade is:  100.0
Your Sorted Grade List is:
100.0
99.0
97.0
95.0
93.0
```

Python Tutorial 13

```
numgrades = int (input('How Many Grades Do You Have? '))
j = 1
i = 0
grades = []
while(j<=numgrades):
    grade = float(input('Please Enter Your Grade: '))
    grades.append(grade)
    print(grades)
    j=j+1
while(i<numgrades):
    print(grades[i])
    i = i+1
```

```
[99.0, 98.0]
Please Enter Your Grade: 97
[99.0, 98.0, 97.0]
99.0
98.0
97.0
```

Python Tutorial 14

```
import pickle
fruits = ['apples','oranges','bananas']
x = 7
y = 3.14
nuts = ['pecans','almond']
grades = [99,100,56,77,85]
dataSet = [fruits,x,y,nuts,grades]
with open('myData.pk1','wb') as f:
    pickle.dump(dataSet,f)
with open('myData.pk1','rb') as f2:
    bigKahuna = pickle.load(f2)
for dt in bigKahuna:
    print(dt)


>>> %Run -c $EDITOR_CONTENT
['apples', 'oranges', 'bananas']
7
3.14
['pecans', 'almond']
[99, 100, 56, 77, 85]
```

Python Tutorial 15

```
import pickle
names = []
grades = []
grade = []
numStudents = int(input('How Many Students Do You Have? '))
for j in range(0,numStudents,1):
    name = input('Plese Enter Student Name: ')
    names.append(name)
    prompt = 'Please Enter '+name+"'s grade "
    grade = float(input(prompt))
    grades.append(grade)
with open('studentData.pk1','wb')as dataF:
    pickle.dump(numStudents,dataF)
    pickle.dump(names,dataF)
    pickle.dump(grades,dataF)
with open('studentData.pk1','rb')as readF:
    numStudents = pickle.load(readF)
    numes = pickle.load(readF)
    grades = pickle.load(readF)
while (1==1):
    name = input('Which Student Do You Want To Check? ')
    for i in range(0,numStudents,1):
        if (names[i] == name):
            print(str(name),"'s grade is "+str(grades[i])+'.'


>>> %Run -c $EDITOR_CONTENT
```

```
How Many Students Do You Have? 3
Plese Enter Student Name: a
Please Enter a's grade 1
Plese Enter Student Name: b
Please Enter b's grade 2
Plese Enter Student Name: c
Please Enter c's grade 3
Which Student Do You Want To Check? b
b 's grade is 2.0.
Which Student Do You Want To Check? a
a 's grade is 1.0.
Which Student Do You Want To Check? c
c 's grade is 3.0.
Which Student Do You Want To Check?
```

Python Tutorial 16

```
num = []
def numgrades(num):
    num = int(input('How Many Grade Do Your Have: '))
    return num
def grade(nums):
    grade = []
    grades = []
    for i in range(0,nums,1):
        grade = float(input('Please Enter Your Grade: '))
        grades.append(grade)
    return grades
def Average(nums,grades):
    Grades = 0.0
    for i in range(0,nums,1):
        Grades = Grades+grades[i]
    average = Grades/nums
    return average
def lowgrade(grades):
    lowGrade = 100
    for i in range(0,nums,1):
        if grades[i]<lowGrade:
            lowGrade = grades[i]
    return lowGrade
def highgrade(grades):
    highGrade = 0
    for i in range(0,nums,1):
        if grades[i]>highGrade:
            highGrade = grades[i]
    return highGrade

nums = numgrades(num)
print(nums)
grades = grade(nums)
print(grades)
average = Average(nums,grades)
print('Your Average = ',average)
lowGrade = lowgrade(grades)
print('Your lowgrade = ',lowGrade)
highGrade = highgrade(grades)
print('Your highgrade = ',highGrade)
```

```
iPad [iCloud Drive] $
How Many Grade Do Your Have: 3
3
Please Enter Your Grade: 90
Please Enter Your Grade: 95
Please Enter Your Grade: 100
[90.0, 95.0, 100.0]
Your Average =  95.0
Your lowgrade =  90.0
Your highgrade =  100.0
```

Python Tutorial 17

```
def inputGrades(nm):
    grades = []
    for i in range(0,nm,1):
        grd = float (input('Please Enter Your Grade'))
        grades.append(grd)
    return grades
def printGrades(nm,x):
    for i in range(0,nm,1):
        print(x[i])
def averageGrades(nm,x):
    tot = 0
    for i in range(0,nm,1):
        tot = tot+x[i]
    average = tot/nm
    return average
```

```
def highLow(nm,x):
    highG = 0
    lowG = 100
    for i in range(0,nm,1):
        if (x[i]<lowG):
            lowG = x[i]
        if (x[i]>highG):
            highG = x[i]
    return highG,lowG
numGrades = int(input('How Many Grade? '))
myGrades = inputGrades(numGrades)
print('')
print('Your Grades Are: ')
printGrades(numGrades,myGrades)
print('')
avg = averageGrades(numGrades,myGrades)
print('Your Average is: ',round(avg,1))
highG, lowG = highLow(numGrades,myGrades)
print('Your High Grade Is: ',highG)
print('Your Low Grades Is: ',lowG)
```

```
iPad [iCloud Drive] $
How Many Grade? 3
Please Enter Your Grade90
Please Enter Your Grade95
Please Enter Your Grade100
Your Grades Are:
90.0
95.0
100.0
Your Average is:  95.0
Your High Grade Is:  100.0
Your Low Grades Is:  90.0
```

Python Tutorial 18

```
class Student:
    def __init__(self,n,gn):
        self.name = n
        self.gradesnum = gn
    def askg(self):
        grad = []
        self.gra = []
        for i in range(0,self.gradesnum,1):
            grad = float(input('Please Enter '+self.name+"'s Grades: "))
            self.gra.append(grad)
        return(self.gra)
    def average(self):
        Grades = 0.00
        for i in range(0,self.gradesnum,1):
            Grades = Grades + self.gra[i]
        average = Grades/self.gradesnum
        return average
    def highg(self):
        highg = 0
        for i in range(0,self.gradesnum,1):
            if(self.gra[i]>highg):
                highg = self.gra[i]
        return highg
    def lowg(self):
        lowg = 100
        for i in range(0,self.gradesnum,1):
            if(self.gra[i]<lowg):
                lowg = self.gra[i]
        return lowg
name1 = Student('a',5)
nameg1 = name1.askg()
print(nameg1)
name2 = Student('b',5)
nameg2 = name2.askg()
print(nameg2)
name3 = Student('c',5)
nameg3 = name3.askg()
print(nameg3)
names = [name1,name2,name3]
namenum = 3
while True:
    sname = input('Which Student Do You Want To Check?')
    for i in range(0,namenum,1):
        if(names[i].name == sname):
            print('name:',names[i].name)
            print('Average = ',names[i].average())
            print('Highgrade = ',names[i].highg())
            print('Lowgrade = ',names[i].lowg())
```

```
iPad [iCloud Drive] $
```

```
Please Enter a's Grades: 98
Please Enter a's Grades: 69
Please Enter a's Grades: 79
Please Enter a's Grades: 68
Please Enter a's Grades: 95
[98.0, 69.0, 79.0, 68.0, 95.0]
Please Enter b's Grades: 57
Please Enter b's Grades: 96
Please Enter b's Grades: 87
Please Enter b's Grades: 56
Please Enter b's Grades: 79
[57.0, 96.0, 87.0, 56.0, 79.0]
Please Enter c's Grades: 98
Please Enter c's Grades: 96
Please Enter c's Grades: 89
Please Enter c's Grades: 79
Please Enter c's Grades: 99
[98.0, 96.0, 89.0, 79.0, 99.0]
Which Student Do You Want To Check?c
name: c
Average =  92.2
Highgrade =  99.0
Lowgrade =  79.0
Which Student Do You Want To Check?a
name: a
Average =  81.8
Highgrade =  98.0
Lowgrade =  68.0
Which Student Do You Want To Check?b
name: b
Average =  75.0
Highgrade =  96.0
Lowgrade =  56.0
Which Student Do You Want To Check?
```

Python Tutorial 19

```
class Students:
    def __init__(self,first,last):
        self.first = first
        self.last = last
    def gInput(self,ng):
        self.ng=ng
        self.grades = []
        for i in range(0,self.ng,1):
            grd = float(input('Please Enter '+self.first+"'s Grade: "))
            self.grades.append(grd)
        return self.grades
    def printGrades(self):
        print(self.first,self.last,"'s Grades are: ")
        for i in range(0,self.ng,1):
            print(self.grades[i])
        print('')
    def avGrades(self):
        bucket = 0
        for i in range(0,self.ng,1):
            bucket = bucket+self.grades[i]
        avg = bucket/self.ng
        return avg
    def highLow(self):
        highGrade = 0
        lowGrade = 100
        for i in range(0,self.ng,1):
            if self.grades[i]>highGrade:
                highGrade = self.grades[i]
            if self.grades[i]<lowGrade:
                lowGrade = self.grades[i]
        return lowGrade,highGrade
student1 = Students('Joe','Evans')
student1.gInput(4)
student2 = Students('Shirly','Baker')
student2.gInput(6)
print('Grade Report for ',student1.first,student1.last)
student1.printGrades()
print('Average is: ',student1.avGrades())
lowGrade,highGrade = student1.highLow()
print('Low Grade is: ',lowGrade)
print('High Grade is: ',highGrade)
print('Grade Report for ',student2.first,student1.last)
student2.printGrades()
print('Average is: ',student2.avGrades())
lowGrade,highGrade = student2.highLow()
print('Low Grade is: ',lowGrade)
print('High Grade is: ',highGrade)
```

```
>>> %Run -c $EDITOR_CONTENT
Please Enter Joe's Grade: 95
Please Enter Joe's Grade: 78
Please Enter Joe's Grade: 96
Please Enter Joe's Grade: 92
Please Enter Shirly's Grade: 76
Please Enter Shirly's Grade: 97
Please Enter Shirly's Grade: 95
Please Enter Shirly's Grade: 86
Please Enter Shirly's Grade: 91
Please Enter Shirly's Grade: 92
Grade Report for  Joe Evans
Joe Evans 's Grades are:
95.0
78.0
96.0
92.0

Average is:  90.25
Low Grade is:  78.0
High Grade is:  96.0
Grade Report for  Shirly Evans
Shirly Baker 's Grades are:
76.0
97.0
95.0
86.0
91.0
92.0

Average is:  89.5
Low Grade is:  76.0
High Grade is:  97.0
```

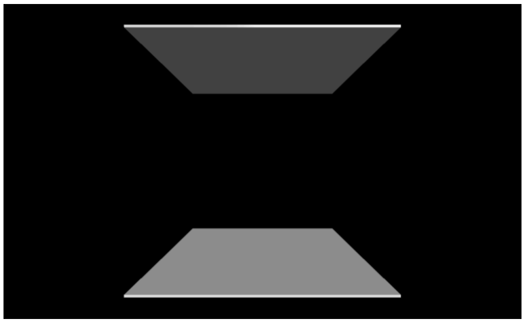
Python Tutorial 20

```
from time import*
from threading import Thread

def myBox(delayT,color):
    while True:
        print('...My Box is Open')
        sleep(delayT)
        print('...My Box is Closed')
        sleep(delayT)
def myLED(delayT,color):
    while True:
        print('My LED is On')
        sleep(delayT)
        print('My LED is Off')
        sleep(delayT)
delayBox = 5
delayLED = 1
boxThread = Thread(target=myBox, args=(delayBox,'red'))
LEDThread = Thread(target=myLED, args=(delayLED,'blue'))
boxThread.daemon=True
LEDThread.daemon=True
boxThread.start()
LEDThread.start()
while True:
    pass
```

```
iPad [iCloud Drive] $
...My Box is Open
My LED is On
My LED is Off
My LED is On
My LED is Off
My LED is On
...My Box is Closed
My LED is Off
My LED is On
```

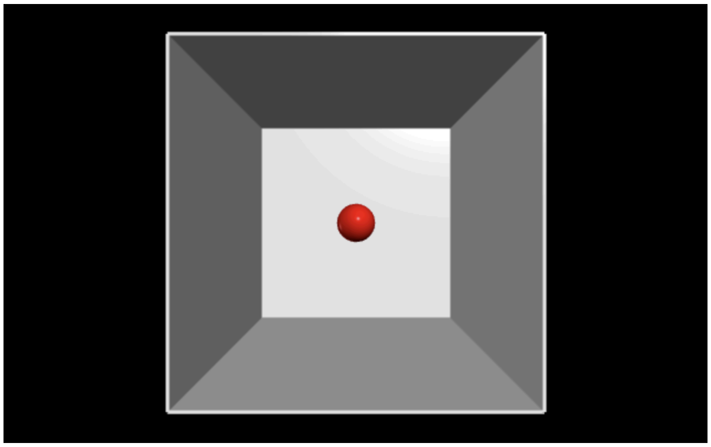
Python 3D Graphics Tutorial 1

```
from vpython import *
from time import *
floor = box(pos = vector(0,-5,0),color = color.white, length = 10,height = .1,width = 10)
ceiling  = box(pos = vector(0,5,0),color = color.white, length = 10,height = .1,width = 10)

while True:
   pass
```



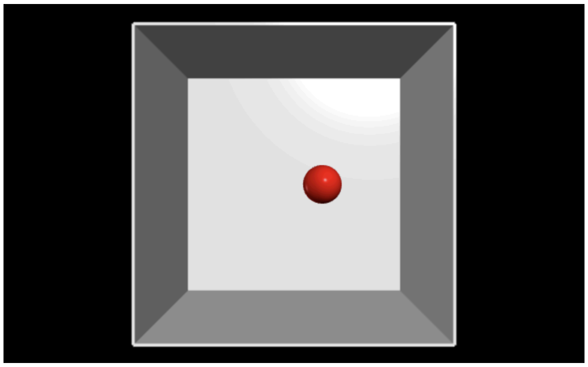
Python 3D Graphics Tutorial 2

```
from vpython import *
from time import *
floor = box(pos = vector(0,-5,0),color = color.white,size = vector(10,.1,10))
ceiling  = box(pos = vector(0,5,0),color = color.white,size = vector(10,.1,10))
backwall = box(pos = vector(0,0,-5),size = vector(10,10,.1),color = color.white)
leftwall = box(pos = vector(-5,0,0),size = vector(.1,10,10),color = color.white)
rightwall = box(pos = vector(5,0,0),size = vector(.1,10,10),color = color.white)
marble = sphere(radius = .75,color = color.red)
deltaX = .1
xPos = 0
while True:
  rate(50)
  xPos = xPos + deltaX
  if (xPos > 5 or xPos < -5):
     deltaX = deltaX*(-1)
  marble.pos = vector(xPos,0,0)
```



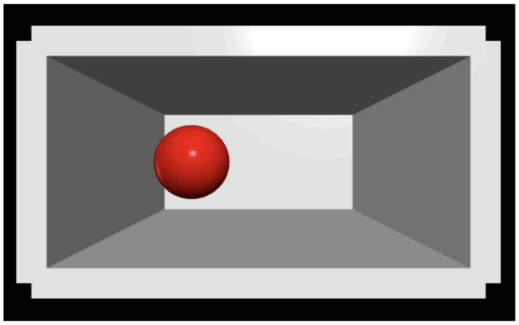
Python 3D Graphics Tutorial 3

```
from vpython import *
from time import *
mRadius = .75
wallThickness = .1
roomWidth = 10
roomDepth = 5
roomHeight = 10
floor = box(pos = vector(0,-roomHeight/2,0),color = color.white,size =
vector(roomWidth,wallThickness,roomDepth))
ceiling  = box(pos = vector(0,roomHeight/2.0,0),color = color.white,size =
vector(roomWidth,wallThickness,roomDepth))
backwall = box(pos = vector(0,0,-roomDepth/2),size =
vector(roomWidth,roomHeight,wallThickness),color = color.white)
leftwall = box(pos = vector(-roomWidth/2,0,0),size =
vector(wallThickness,roomHeight,roomDepth),color = color.white)
rightwall = box(pos = vector(roomWidth/2,0,0),size =
vector(wallThickness,roomHeight,roomDepth),color = color.white)
marble = sphere(radius = mRadius,color = color.red)
deltaX = .1
xPos = 0
while True:
  rate(10)
  xPos = xPos + deltaX
  if (xPos > roomWidth/2 or xPos <3- roomWidth/2):
     deltaX = deltaX*(-1)
  marble.pos = vector(xPos,0,0)
```



Python 3D Graphics Tutorial 4

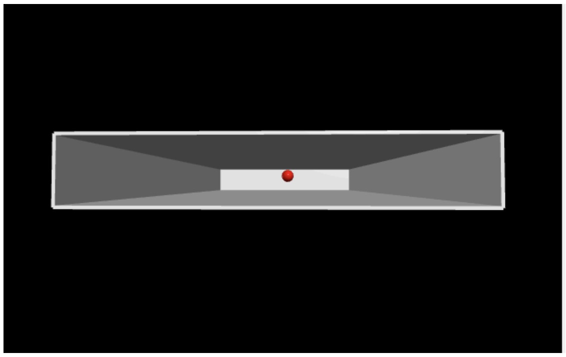
```
from vpython import *
from time import *
mRadius = 2
wallThickness = 1
roomWidth = 15
roomDepth = 12
roomHeight = 8
floor = box(pos = vector(0,-roomHeight/2,0),color = color.white,size =
vector(roomWidth,wallThickness,roomDepth))
ceiling  = box(pos = vector(0,roomHeight/2.0,0),color = color.white,size =
vector(roomWidth,wallThickness,roomDepth))
backwall = box(pos = vector(0,0,-roomDepth/2),size =
vector(roomWidth,roomHeight,wallThickness),color = color.white)
leftwall = box(pos = vector(-roomWidth/2,0,0),size =
vector(wallThickness,roomHeight,roomDepth),color = color.white)
rightwall = box(pos = vector(roomWidth/2,0,0),size =
vector(wallThickness,roomHeight,roomDepth),color = color.white)
marble = sphere(radius = mRadius,color = color.red)
deltaX = .1
xPos = 0
while True:
   rate(10)
   xPos = xPos + deltaX
   Xrme = xPos + mRadius
   Xlme = xPos - mRadius
   Rwe = roomWidth/2 - wallThickness/2
   Lwe =- roomWidth/2 + wallThickness/2
   if (Xrme >= Rwe or Xlme < Lwe):
       deltaX = deltaX*(-1)
   marble.pos = vector(xPos,0,0)
```



Python 3D Graphics Tutorial 5

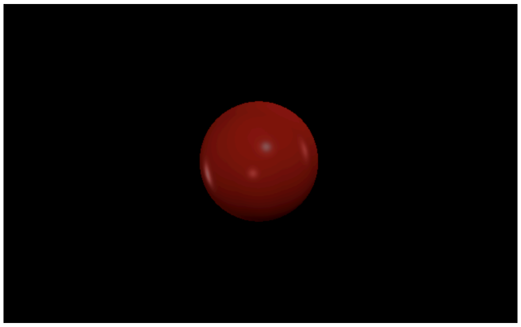
```
from vpython import *
from time import *
mRadius = .5
wallThickness = .1
roomWidth = 12
roomDepth = 20
roomHeight = 2
floor = box(pos = vector(0,-roomHeight/2,0),color = color.white,size =
vector(roomWidth,wallThickness,roomDepth))
ceiling  = box(pos = vector(0,roomHeight/2.0,0),color = color.white,size =
vector(roomWidth,wallThickness,roomDepth))
backwall = box(pos = vector(0,0,-roomDepth/2),size =
vector(roomWidth,roomHeight,wallThickness),color = color.white)
leftwall = box(pos = vector(-roomWidth/2,0,0),size =
vector(wallThickness,roomHeight,roomDepth),color = color.white)
rightwall = box(pos = vector(roomWidth/2,0,0),size =
vector(wallThickness,roomHeight,roomDepth),color = color.white)
marble = sphere(radius = mRadius,color = color.red)
deltaX = .1
deltaY = .1
deltaZ = .1

xPos = 0
```

```
yPos = 0
zPos = 0

while True:
  rate(25)
  xPos = xPos + deltaX
  yPos = yPos + deltaY
  zPos = zPos + deltaZ
  Xrme = xPos + mRadius
  Xlme = xPos - mRadius
  Ytme = yPos + mRadius
  Ybme = yPos - mRadius
  Zbme = zPos - mRadius
  Zfme = zPos + mRadius
  Rwe = roomWidth/2 - wallThickness/2
  Lwe = -roomWidth/2 + wallThickness/2
  Cwe = roomHeight/2 - wallThickness/2
  Floorwe = -roomHeight/2 + wallThickness/2
  Bwe = -roomDepth/2 + wallThickness/2
  Fwe = roomDepth/2 - wallThickness/2
  if (Xrme >= Rwe or Xlme <= Lwe):
     deltaX = deltaX*(-1)
  if (Ytme >= Cwe or Ybme <= Floorwe):
     deltaY = deltaY*(-1)
  if (Zfme >= Fwe or Zbme <= Bwe):
     deltaZ = deltaZ*(-1)
  marble.pos = vector(xPos,yPos,zPos)
```



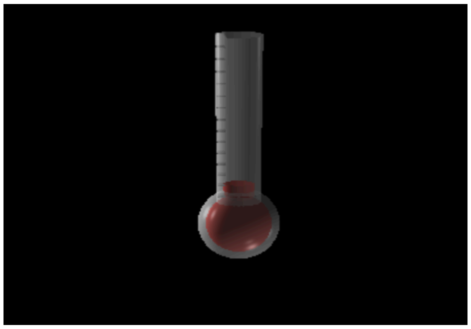
Python 3D Graphics Tutorial 6

```
from vpython import*
import numpy as np
Piston1 = sphere(radius = 1,color = color.red,opacity = .5)
while True:
    for myRadius in np.linspace(.1,1,1000):
        rate(250)
        Piston1.radius = myRadius
    for myRadius in np.linspace(1,.1,1000):
        rate(250)
        Piston1.radius = myRadius
```



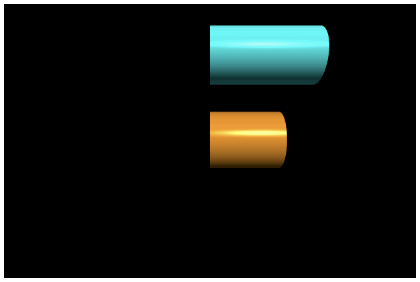
Python 3D Graphics Tutorial 7

```
from vpython import*
import numpy as np
glassBulb = sphere(radius = 1.25,color = color.white,opacity = .5)
glassCyl = cylinder(radius = .65,length = 6,color = color.white,opacity = .5)
mercSphere = sphere(radius = 1,color = color.red)
mercColumn = cylinder(radius = .45,length = 6,color = color.red)
for tick in np.linspace(1,6,15):
    box(size = vector(.05,.5,.25),color = color.white,pos = vector(tick,0,.5))
while True:
    for myTemp in np.linspace(1,6,100):
        rate(100)
        mercColumn.length = myTemp
    for myTemp in np.linspace(6,1,100):
        rate(100)
        mercColumn.length = myTemp
```



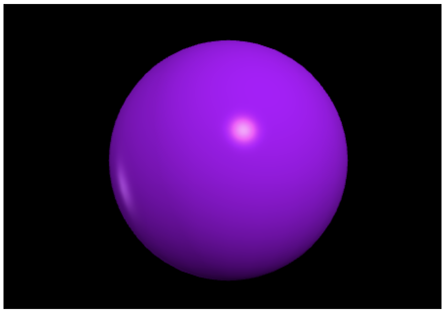
Python 3D Graphics Tutorial 8

```
from vpython import*
myCylOrange = cylinder(radius = 1,color = color.orange,length = 6)
myCylCyan = cylinder(radius = 1, length = 6, color = color.cyan,pos = vector(0,3,0))
myCylOrangeLength = 1
myCylCyanLength = 1
myCylOrangeDelta = .01
myCylCyanDelta = .02
while True:
    rate(50)
    myCylOrangeLength = myCylOrangeLength + myCylOrangeDelta
    myCylCyanLength = myCylCyanLength + myCylCyanDelta
    myCylOrange.length = myCylOrangeLength
    myCylCyan.length = myCylCyanLength
    if myCylOrangeLength >= 6 or myCylOrangeLength <= .1:
        myCylOrangeDelta = myCylOrangeDelta*(-1)
    if myCylCyanLength >= 6 or myCylCyanLength <= .1:
        myCylCyanDelta = myCylCyanDelta*(-1)
```



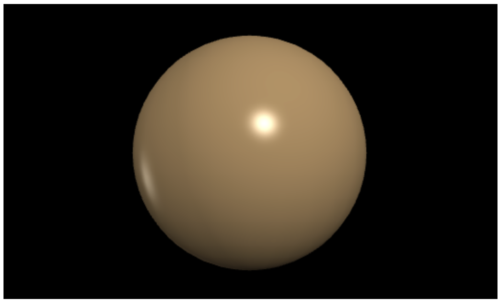
Python 3D Graphics Tutorial 9

```
from vpython import*
mySphere = sphere(radius = 1,color = vector(.7,0,1))
while True:
  pass
```



Python 3D Graphics Tutorial 10

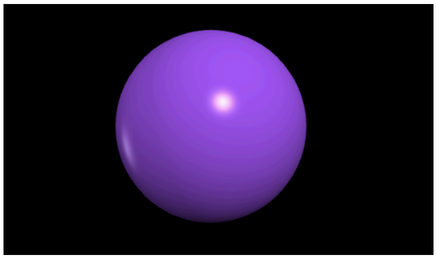
```
from vpython import*
myOrb = sphere(radius = 1,color = color.white)
rChan = 0
gChan = 0
bChan = 0
rInc = .001
gInc = .002
bInc = .015
while True:
  rate(500)
  rChan = rChan + rInc
  gChan = gChan + gInc
  bChan = bChan + bInc
  myOrb.color = vector(rChan,gChan,bChan)
  if rChan >= 1 or rChan <= 0:
    rInc = rInc*(-1)
  if gChan >= 1 or gChan <= 0:
    gInc = gInc*(-1)
  if bChan >= 1 or bChan <= 0:
    bInc = rInc*(-1)
```



Python 3D Graphics Tutorial 11

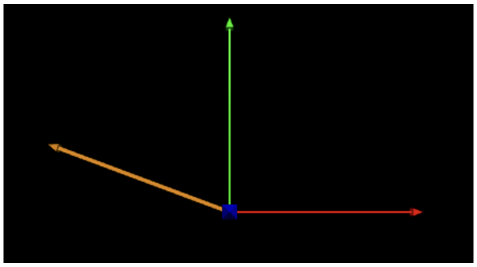
```
from vpython import *
myOrb = sphere(radius = 1,color = vector(1,1,0))
rChan = 1
gChan = 1
bChan = 0
rInc = .001
gInc = -.001
bInc = .001
while True:
  rate(100)
  rChan = rChan + rInc
  gChan = gChan + gInc
```

```
bChan = bChan + bInc

if rChan <= 1:
    rApply = rChan
if rChan > 1:
    rApply = 1

if gChan <= 1:
    gApply = gChan
if gChan > 1:
    gApply = 1

if bChan <= 1:
    bApply = bChan
if bChan > 1:
    bApply = 1

myOrb.color = vector(rApply,gApply,bApply)

if rChan >= 1.5 or rChan <= 0:
    rInc = rInc*(-1)

if gChan >= 1.5 or gChan <= 0:
    gInc = gInc*(-1)

if bChan >= 1.5 or bChan <= 0:
    bInc = bInc*(-1)

print(rApply + gApply + bApply)
```



Python 3D Graphics Tutorial 12

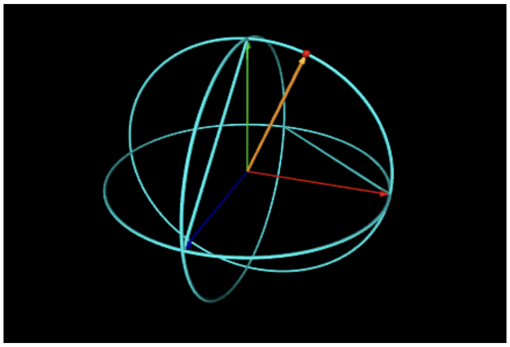
```
from vpython import*
import numpy as np
arrowL = 2
arrowT = .02
pntT = .04
theta = 0
Xarrow = arrow(axis = vector(1,0,0),color = color.red,length = arrowL,shaftwidth = arrowT)
Yarrow = arrow(axis = vector(0,1,0),color = color.green,length = arrowL,shaftwidth = arrowT)
Zarrow = arrow(axis = vector(0,0,1),color = color.blue,length = arrowL,shaftwidth = arrowT)
pntArrow = arrow(axis = vector(arrowL*np.cos(theta),arrowL*np.sin(theta),0),color =
color.orange,length = arrowL,shaftwidth = pntT)
while True:
  for myAngle in np.linspace(0,2*np.pi,1000):
      rate(50)
      pntArrow.axis = vector(arrowL*np.cos(myAngle),arrowL*np.sin(myAngle),0)
      pntArrow.length = arrowL
```



Python 3D Graphics Tutorial 13

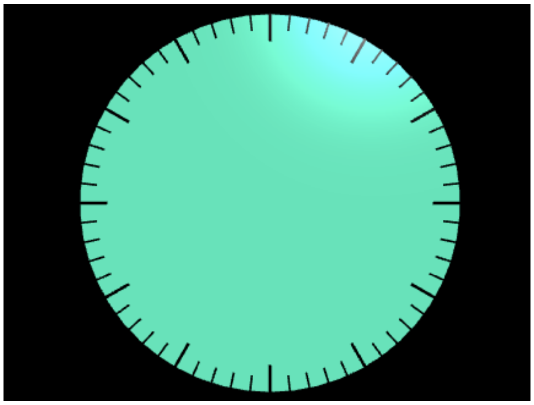
```
from vpython import *
import numpy as np
arrowL = 2
arrowT = .02
pntT = .04
bRadius = .05
```

```
Xarrow = arrow(axis = vector(1,0,0),color = color.red,length = arrowL,shaftwidth = arrowT)
Yarrow = arrow(axis = vector(0,1,0),color = color.green,length = arrowL,shaftwidth = arrowT)
Zarrow = arrow(axis = vector(0,0,1),color = color.blue,length = arrowL,shaftwidth = arrowT)
Parrow = arrow(axis = vector(1,0,0),color = color.orange,length = arrowL,shaftwidth = pntT)
myBall = sphere(make_trail = True,trail_color = color.cyan,radius = bRadius,color =
color.red,pos = vector(arrowL,0,0))
while True:
  for myAngle in np.linspace(0,2*np.pi,1000):
     rate(50)
     Parrow.axis = vector(arrowL*np.cos(myAngle),arrowL*np.sin(myAngle),0)
     Parrow.length = arrowL
     myBall.pos = vector(arrowL*np.cos(myAngle),arrowL*np.sin(myAngle),0)
  for myAngle in np.linspace(0,5*np.pi/2,1000):
     rate(50)
     Parrow.axis = vector(arrowL*np.cos(myAngle),0,arrowL*np.sin(myAngle))
     Parrow.length = arrowL
     myBall.pos = vector(arrowL*np.cos(myAngle),0,arrowL*np.sin(myAngle))
  for myAngle in np.linspace(0,2*np.pi,1000):
     rate(50)
     Parrow.axis = vector(0,arrowL*np.sin(myAngle),arrowL*np.cos(myAngle))
     Parrow.length = arrowL
     myBall.pos = vector(0,arrowL*np.sin(myAngle),arrowL*np.cos(myAngle))
  for myAngle in np.linspace(np.pi/2,2*np.pi/2,1000):
     rate(50)
     Parrow.axis = vector(0,arrowL*np.sin(myAngle),arrowL*np.cos(myAngle))
     Parrow.length = arrowL
     myBall.pos = vector(0,arrowL*np.sin(myAngle),arrowL*np.cos(myAngle))
```



Python 3D Graphics Tutorial 14

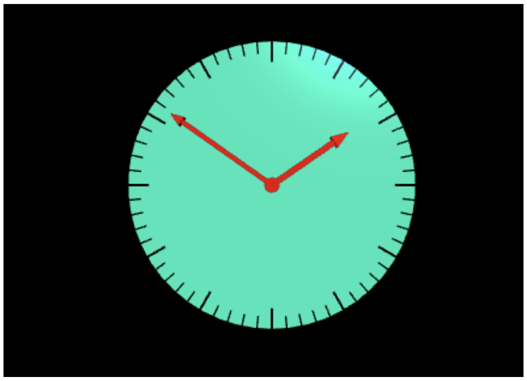
```
from vpython import *
import numpy as np
clockR = 2
clockT = clockR/10
majorTickL = clockR/7
majorTickT = 2*np.pi*clockR/400
majorTickW = clockT*1.2
manorTickL = clockR/12
manorTickT = 2*np.pi*clockR/600
manorTickW = clockT*1.2
for theta in np.linspace(0,2*np.pi,13):
    majorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),color =
color.black,length = majorTickL,width = majorTickW,height = majorTickT,pos =
vector((clockR-majorTickL/2)*np.cos(theta),(clockR-majorTickL/2)*np.sin(theta),0))
for theta in np.linspace(0,2*np.pi,61):
    manorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),color =
color.black,length = manorTickL,width = manorTickW,height = manorTickT,pos =
vector((clockR-manorTickL/2)*np.cos(theta),(clockR-manorTickL/2)*np.sin(theta),0))
clockFace = cylinder(axis = vector(0,0,1),color = vector(0,1,.8),length = clockT,radius =
clockR,pos = vector(0,0,-clockT/2))
while True:
    pass
```



Python 3D Graphics Tutorial 15

```
from vpython import *
import numpy as np
clockR = 2
clockT = clockR/10
majorTickL = clockR/7
majorTickT = 2*np.pi*clockR/400
majorTickW = clockT*1.2
manorTickL = clockR/12
manorTickT = 2*np.pi*clockR/600
manorTickW = clockT*1.2
minuteHandL = clockR-majorTickL
minuteHandT = minuteHandL/25
mniuteHandOffset = clockT/2 + minuteHandT
hubRadius = clockT/2
hourHandL = .75*minuteHandL
hourHandT = minuteHandT*1.25
hourHandOffset = clockT/2 + hourHandT
hourRadius = clockT/2
hourAngle = np.pi/2
minuteAngle = np.pi/2
minInc = .01
hourInc = minInc/12
for theta in np.linspace(0,2*np.pi,13):
   majorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),color = color.black,length = majorTickL,width = majorTickW,height = majorTickT,pos = vector((clockR-majorTickL/2)*np.cos(theta),(clockR-majorTickL/2)*np.sin(theta),0))
for theta in np.linspace(0,2*np.pi,61):
   manorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),color = color.black,length = manorTickL,width = manorTickW,height = manorTickT,pos = vector((clockR-manorTickL/2)*np.cos(theta),(clockR-manorTickL/2)*np.sin(theta),0))
```

```
clockFace = cylinder(axis = vector(0,0,1),color = vector(0,1,.8),length = clockT,radius = clockR,pos = vector(0,0,-clockT/2))
minuteHand = arrow(axis = vector(1,0,0),color = color.red,shaftwidth = minuteHandT,length = minuteHandL,pos = vector(0,0,mniuteHandOffset))
hourHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = hourHandT,length = hourHandL,pos = vector(0,0,hourHandOffset))
hub = cylinder(axis = vector(0,0,1),color = color.red,radius = hubRadius,length = 2*clockT)
while True:
  rate(50)
  hourAngle = hourAngle - hourInc
  minuteAngle = minuteAngle - minInc
  hourHand.axis = vector(hourHandL*np.cos(hourAngle),hourHandL*np.sin(hourAngle),0)
  minuteHand.axis = vector(minuteHandL*np.cos(minuteAngle),minuteHandL*np.sin(minuteAngle),0)
```

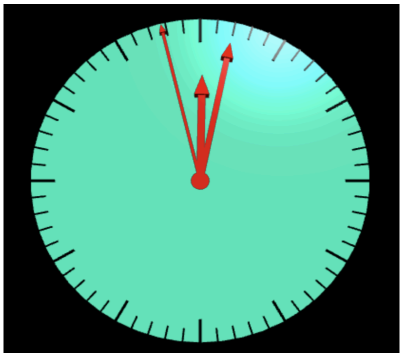


Python 3D Graphics Tutorial 16

```
from vpython import *
import numpy as np
clockR = 2
clockT = clockR/10
majorTickL = clockR/7
majorTickT = 2*np.pi*clockR/400
majorTickW = clockT*1.2
manorTickL = clockR/12
```

```
manorTickT = 2*np.pi*clockR/600
manorTickW = clockT*1.2
minuteHandL = clockR-majorTickL
minuteHandT = minuteHandL/25
mniuteHandOffset = clockT/2 + minuteHandT
hubRadius = clockT/2
hourHandL = .75*minuteHandL
hourHandT = minuteHandT*1.25
hourHandOffset = clockT/2 + hourHandT
hourRadius = clockT/2
hourAngle = np.pi/2
minuteAngle = np.pi/2
minInc = .0001
hourInc = minInc/12
secondHandL = clockR - majorTickL/2
secondHandT = minuteHandL/50
secondHandOffset = clockT*1.5 + minuteHandT
secondAngle = np.pi/2
secondInc = minInc*60

for theta in np.linspace(0,2*np.pi,13):
  majorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),color =
color.black,length = majorTickL,width = majorTickW,height = majorTickT,pos =
vector((clockR-majorTickL/2)*np.cos(theta),(clockR-majorTickL/2)*np.sin(theta),0))
for theta in np.linspace(0,2*np.pi,61):
  manorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),color =
color.black,length = manorTickL,width = manorTickW,height = manorTickT,pos =
vector((clockR-manorTickL/2)*np.cos(theta),(clockR-manorTickL/2)*np.sin(theta),0))
clockFace = cylinder(axis = vector(0,0,1),color = vector(0,1,.8),length = clockT,radius =
clockR,pos = vector(0,0,-clockT/2))
minuteHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = minuteHandT,length =
minuteHandL,pos = vector(0,0,mniuteHandOffset))
hourHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = hourHandT,length =
hourHandL,pos = vector(0,0,hourHandOffset))
```

```
hub = cylinder(axis = vector(0,0,1),color = color.red,radius = hubRadius,length = 2*clockT)
secondHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = secondHandT,length =
secondHandL,pos = vector(0,0,secondHandOffset))
while True:
  rate(18)
  hourAngle = hourAngle - hourInc
  minuteAngle = minuteAngle - minInc
  secondAngle = secondAngle - secondInc
  hourHand.axis = vector(hourHandL*np.cos(hourAngle),hourHandL*np.sin(hourAngle),0)
  minuteHand.axis =
vector(minuteHandL*np.cos(minuteAngle),minuteHandL*np.sin(minuteAngle),0)
  secondHand.axis =
vector(secondHandL*np.cos(secondAngle),secondHandL*np.sin(secondAngle),0)
```

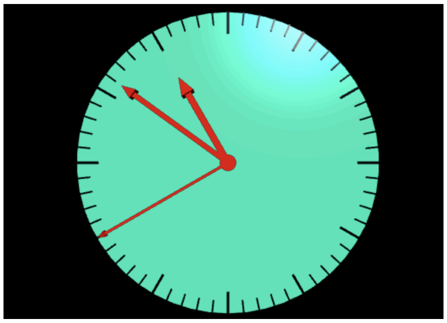


Python 3D Graphics Tutorial 17

```
from vpython import *
import numpy as np
import time
clockR = 2
clockT = clockR/10
majorTickL = clockR/7
majorTickT = 2*np.pi*clockR/400
majorTickW = clockT*1.2
manorTickL = clockR/12
manorTickT = 2*np.pi*clockR/600
```

```
manorTickW = clockT*1.2
minuteHandL = clockR-majorTickL
minuteHandT = minuteHandL/25
mniuteHandOffset = clockT/2 + minuteHandT
hubRadius = clockT/2
hourHandL = .75*minuteHandL
hourHandT = minuteHandT*1.25
hourHandOffset = clockT/2 + hourHandT
hourRadius = clockT/2
hourAngle = np.pi/2
minuteAngle = np.pi/2
minInc = .0001
hourInc = minInc/12
secondHandL = clockR - majorTickL/2
secondHandT = minuteHandL/50
secondHandOffset = clockT*1.5 + minuteHandT
secondAngle = np.pi/2
secondInc = minInc*60
for theta in np.linspace(0,2*np.pi,13):
   majorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),color =
color.black,length = majorTickL,width = majorTickW,height = majorTickT,pos =
vector((clockR-majorTickL/2)*np.cos(theta),(clockR-majorTickL/2)*np.sin(theta),0))
for theta in np.linspace(0,2*np.pi,61):
   manorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),color =
color.black,length = manorTickL,width = manorTickW,height = manorTickT,pos =
vector((clockR-manorTickL/2)*np.cos(theta),(clockR-manorTickL/2)*np.sin(theta),0))
clockFace = cylinder(axis = vector(0,0,1),color = vector(0,1,.8),length = clockT,radius =
clockR,pos = vector(0,0,-clockT/2))
minuteHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = minuteHandT,length =
minuteHandL,pos = vector(0,0,mniuteHandOffset))
hourHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = hourHandT,length =
hourHandL,pos = vector(0,0,hourHandOffset))
hub = cylinder(axis = vector(0,0,1),color = color.red,radius = hubRadius,length = 2*clockT)
```

```
secondHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = secondHandT,length =
secondHandL,pos = vector(0,0,secondHandOffset))
while True:
  rate(5000)
  hour = time.localtime(time.time())[3]
  if hour>12:
     hour - hour-12
  minute = time.localtime(time.time())[4]
  second = time.localtime(time.time())[5]
  hourAngle = -(hour/12)*2*np.pi + np.pi/2
  minuteAngle = -(minute/60)*2*np.pi + np.pi/2
  secondAngle = -(second/60)*2*np.pi + np.pi/2
  print(second)
  hourHand.axis = vector(hourHandL*np.cos(hourAngle),hourHandL*np.sin(hourAngle),0)
  minuteHand.axis =
vector(minuteHandL*np.cos(minuteAngle),minuteHandL*np.sin(minuteAngle),0)
  secondHand.axis =
vector(secondHandL*np.cos(secondAngle),secondHandL*np.sin(secondAngle),0)
```

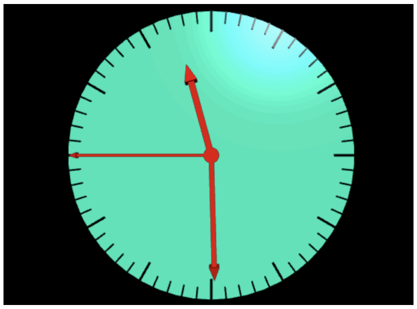


Python 3D Graphics Tutorial 18

```
from vpython import *
import numpy as np
import time
clockR = 2
clockT = clockR/10
majorTickL = clockR/7
```

```
majorTickT = 2*np.pi*clockR/400
majorTickW = clockT*1.2
manorTickL = clockR/12
manorTickT = 2*np.pi*clockR/600
manorTickW = clockT*1.2
minuteHandL = clockR-majorTickL
minuteHandT = minuteHandL/25
mniuteHandOffset = clockT/2 + minuteHandT
hubRadius = clockT/2
hourHandL = .75*minuteHandL
hourHandT = minuteHandT*1.25
hourHandOffset = clockT/2 + hourHandT
hourRadius = clockT/2
hourAngle = np.pi/2
minuteAngle = np.pi/2
minInc = .0001
hourInc = minInc/12
secondHandL = clockR - majorTickL/2
secondHandT = minuteHandL/50
secondHandOffset = clockT*1.5 + minuteHandT
secondAngle = np.pi/2
secondInc = minInc*60
for theta in np.linspace(0,2*np.pi,13):
   majorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),
                   color = color.black,length = majorTickL,width = majorTickW,
                   height = majorTickT,pos =
vector((clockR-majorTickL/2)*np.cos(theta),(clockR-majorTickL/2)*np.sin(theta),0))
for theta in np.linspace(0,2*np.pi,61):
   manorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),
                   color = color.black,length = manorTickL,width = manorTickW,height = manorTickT,
                   pos =
vector((clockR-manorTickL/2)*np.cos(theta),(clockR-manorTickL/2)*np.sin(theta),0))
clockFace = cylinder(axis = vector(0,0,1),color = vector(0,1,.8),length = clockT,radius =
clockR,pos = vector(0,0,-clockT/2))
```

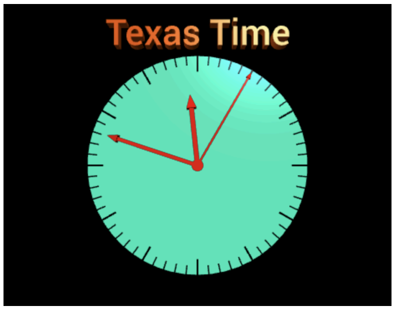
```
minuteHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = minuteHandT,
            length = minuteHandL,pos = vector(0,0,mniuteHandOffset))
hourHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = hourHandT,
           length = hourHandL,pos = vector(0,0,hourHandOffset))
hub = cylinder(axis = vector(0,0,1),color = color.red,radius = hubRadius,length = 2*clockT)
secondHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = secondHandT,
            length = secondHandL,pos = vector(0,0,secondHandOffset))
while True:
    rate(5000)
    hour = time.localtime(time.time())[3]
    if hour>12:
        hour - hour-12
    minute = time.localtime(time.time())[4]
    second = time.localtime(time.time())[5]
    hourAngle = -((hour+minute/60)/12)*2*np.pi + np.pi/2
    minuteAngle = -((minute+second/60)/60)*2*np.pi + np.pi/2
    secondAngle = -(second/60)*2*np.pi + np.pi/2
    print(second)
    hourHand.axis = vector(hourHandL*np.cos(hourAngle),hourHandL*np.sin(hourAngle),0)
    minuteHand.axis =
vector(minuteHandL*np.cos(minuteAngle),minuteHandL*np.sin(minuteAngle),0)
    secondHand.axis =
vector(secondHandL*np.cos(secondAngle),secondHandL*np.sin(secondAngle),0)
```



Python 3D Graphics Tutorial 19

```
from vpython import *
```

```
import numpy as np
import time
clockR = 2
clockT = clockR/10
majorTickL = clockR/7
majorTickT = 2*np.pi*clockR/400
majorTickW = clockT*1.2
manorTickL = clockR/12
manorTickT = 2*np.pi*clockR/600
manorTickW = clockT*1.2
minuteHandL = clockR-majorTickL
minuteHandT = minuteHandL/25
mniuteHandOffset = clockT/2 + minuteHandT
hubRadius = clockT/2
hourHandL = .75*minuteHandL
hourHandT = minuteHandT*1.25
hourHandOffset = clockT/2 + hourHandT
hourRadius = clockT/2
hourAngle = np.pi/2
minuteAngle = np.pi/2
minInc = .0001
hourInc = minInc/12
secondHandL = clockR - majorTickL/2
secondHandT = minuteHandL/50
secondHandOffset = clockT*1.5 + minuteHandT
secondAngle = np.pi/2
secondInc = minInc*60
for theta in np.linspace(0,2*np.pi,13):
  majorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),
             color = color.black,length = majorTickL,width = majorTickW,
             height = majorTickT,pos =
vector((clockR-majorTickL/2)*np.cos(theta),(clockR-majorTickL/2)*np.sin(theta),0))
for theta in np.linspace(0,2*np.pi,61):
  manorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),
                 color = color.black,length = manorTickL,width = manorTickW,height = manorTickT,
                 pos =
vector((clockR-manorTickL/2)*np.cos(theta),(clockR-manorTickL/2)*np.sin(theta),0))
clockFace = cylinder(axis = vector(0,0,1),color = vector(0,1,.8),length = clockT,radius =
clockR,pos = vector(0,0,-clockT/2))
minuteHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = minuteHandT,
                length = minuteHandL,pos = vector(0,0,mniuteHandOffset))
hourHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = hourHandT,
               length = hourHandL,pos = vector(0,0,hourHandOffset))
hub = cylinder(axis = vector(0,0,1),color = color.red,radius = hubRadius,length = 2*clockT)
secondHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = secondHandT,
                length = secondHandL,pos = vector(0,0,secondHandOffset))
textH = clockR/4
myLabel = text(text = 'Texas Time',align = 'center',color = color.orange,height = textH,pos =
vector(0,1.1*clockR,-clockT/2),depth = clockT)
while True:
  rate(5000)
  hour = time.localtime(time.time())[3]
  if hour>12:
     hour - hour-12
  minute = time.localtime(time.time())[4]
  second = time.localtime(time.time())[5]
  hourAngle = -((hour+minute/60)/12)*2*np.pi + np.pi/2
  minuteAngle = -((minute+second/60)/60)*2*np.pi + np.pi/2
  secondAngle = -(second/60)*2*np.pi + np.pi/2
  print(second)
  hourHand.axis = vector(hourHandL*np.cos(hourAngle),hourHandL*np.sin(hourAngle),0)
  minuteHand.axis =
vector(minuteHandL*np.cos(minuteAngle),minuteHandL*np.sin(minuteAngle),0)
  secondHand.axis =
vector(secondHandL*np.cos(secondAngle),secondHandL*np.sin(secondAngle),0)
```



Python 3D Graphics Tutorial 20

```
from vpython import *
import numpy as np
import time
clockR = 2
clockT = clockR/10
majorTickL = clockR/7
majorTickT = 2*np.pi*clockR/400
majorTickW = clockT*1.2
manorTickL = clockR/12
manorTickT = 2*np.pi*clockR/600
manorTickW = clockT*1.2
minuteHandL = clockR-majorTickL
minuteHandT = minuteHandL/25
mniuteHandOffset = clockT/2 + minuteHandT
hubRadius = clockT/2
hourHandL = .75*minuteHandL
hourHandT = minuteHandT*1.25
hourHandOffset = clockT/2 + hourHandT
hourRadius = clockT/2
hourAngle = np.pi/2
minuteAngle = np.pi/2
minInc = .0001
hourInc = minInc/12
secondHandL = clockR - majorTickL/2
```

```
secondHandT = minuteHandL/50
secondHandOffset = clockT*1.5 + minuteHandT
secondAngle = np.pi/2
secondInc = minInc*60
for theta in np.linspace(0,2*np.pi,13):
  majorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),
            color = color.black,length = majorTickL,width = majorTickW,
            height = majorTickT,pos =
vector((clockR-majorTickL/2)*np.cos(theta),(clockR-majorTickL/2)*np.sin(theta),0))
for theta in np.linspace(0,2*np.pi,61):
  manorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),
            color = color.black,length = manorTickL,width = manorTickW,height = manorTickT,
            pos =
vector((clockR-manorTickL/2)*np.cos(theta),(clockR-manorTickL/2)*np.sin(theta),0))
clockFace = cylinder(axis = vector(0,0,1),color = vector(0,1,.8),length = clockT,radius =
clockR,pos = vector(0,0,-clockT/2))
minuteHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = minuteHandT,
            length = minuteHandL,pos = vector(0,0,mniuteHandOffset))
hourHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = hourHandT,
           length = hourHandL,pos = vector(0,0,hourHandOffset))
hub = cylinder(axis = vector(0,0,1),color = color.red,radius = hubRadius,length = 2*clockT)
secondHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = secondHandT,
            length = secondHandL,pos = vector(0,0,secondHandOffset))
textH = clockR/4
myLabel = text(text = 'Texas Time',align = 'center',color = color.orange,height = textH,pos =
vector(0,1.1*clockR,-clockT/2),depth = clockT)
Angle = np.pi/2
AngleInc = -2*np.pi/12
Angle = Angle + AngleInc
numH = clockR/6
for i in range(1,13,1):
  clockNum = text(align = 'center',text = str(i),pos =
vector(clockR*.75*np.cos(Angle),clockR*.75*np.sin(Angle) - numH/2,0),height = numH,depth =
clockT,color = color.orange)
```

```
  Angle = Angle + AngleInc
while True:
  rate(5000)
  hour = time.localtime(time.time())[3]
  if hour>12:
     hour - hour-12
  minute = time.localtime(time.time())[4]
  second = time.localtime(time.time())[5]
  hourAngle = -((hour+minute/60)/12)*2*np.pi + np.pi/2
  minuteAngle = -((minute+second/60)/60)*2*np.pi + np.pi/2
  secondAngle = -(second/60)*2*np.pi + np.pi/2
  print(second)
  hourHand.axis = vector(hourHandL*np.cos(hourAngle),hourHandL*np.sin(hourAngle),0)
  minuteHand.axis =
vector(minuteHandL*np.cos(minuteAngle),minuteHandL*np.sin(minuteAngle),0)
  secondHand.axis =
vector(secondHandL*np.cos(secondAngle),secondHandL*np.sin(secondAngle),0)
```

Texas Time
12
1
2
3
4
5
6
7
8
9
10
11

Python 3D Graphics Tutorial 21

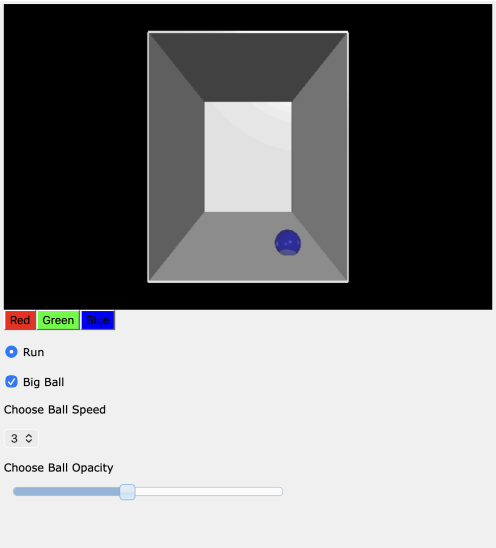
```
from vpython import *
from time import *
mRadius = .5
wallThickness = .1
roomWidth = 12
roomDepth = 20
roomHeight = 15
```

```
floor = box(pos = vector(0,-roomHeight/2,0),color = color.white,size =
vector(roomWidth,wallThickness,roomDepth))
ceiling  = box(pos = vector(0,roomHeight/2.0,0),color = color.white,size =
vector(roomWidth,wallThickness,roomDepth))
backwall = box(pos = vector(0,0,-roomDepth/2),size =
vector(roomWidth,roomHeight,wallThickness),color = color.white)
leftwall = box(pos = vector(-roomWidth/2,0,0),size =
vector(wallThickness,roomHeight,roomDepth),color = color.white)
rightwall = box(pos = vector(roomWidth/2,0,0),size =
vector(wallThickness,roomHeight,roomDepth),color = color.white)
marble = sphere(radius = mRadius,color = color.red)
deltaX = .1
deltaY = .1
deltaZ = .1


xPos = 0
yPos = 0
zPos = 0

run = 0
mySpeed = 1

def ballColorRed(x):
   marble.color = color.red
button(bind = ballColorRed,text = 'Red',color = color.black,background = color.red)
def ballColorGreen(x):
   marble.color = color.green
button(bind = ballColorGreen,text = 'Green',color = color.black,background = color.green)
def ballColorBlue(x):
   marble.color = color.blue
button(bind = ballColorBlue,text = 'Blue',color = color.black,background = color.blue)
scene.append_to_caption('\n\n')
```

```
def runRadio(x):
    print(x.checked)
    global run
    if x.checked == True:
        run = 1
    if x.checked == False:
        run = 0
radio(bind = runRadio, text = 'Run')
scene.append_to_caption('\n\n')

def bigBall(x):
    global mRadius
    if x.checked == True:
        mRadius = mRadius*2
        marble.radius = mRadius
    if x.checked == False:
        mRadius = mRadius/2
        marble.radius = mRadius
checkbox(bind = bigBall,text = 'Big Ball')
scene.append_to_caption('\n\n')
wtext(text = 'Choose Ball Speed')
scene.append_to_caption('\n\n')
def speed(x):
    global mySpeed
    if x.selected == '1':
        mySpeed = 1
    if x.selected == '2':
        mySpeed = 2
    if x.selected == '3':
        mySpeed = 3
    if x.selected == '4':
        mySpeed = 4
    if x.selected == '5':
        mySpeed = 5
```

```
menu(bind = speed,choices = ['1','2','3','4','5'])
scene.append_to_caption('\n\n')
wtext(text = 'Choose Ball Opacity')
scene.append_to_caption('\n\n')
def ballOpacity(x):
   op = x.value
   marble.opacity = op
slider(bind = ballOpacity,vertical = False,min = 0,max = 1,value = 1)

while True:
  rate(25)
  xPos = xPos + deltaX*run*mySpeed
  yPos = yPos + deltaY*run*mySpeed
  zPos = zPos + deltaZ*run*mySpeed


  Xrme = xPos + mRadius
  Xlme = xPos - mRadius
  Ytme = yPos + mRadius
  Ybme = yPos - mRadius
  Zbme = zPos - mRadius
  Zfme = zPos + mRadius
  Rwe = roomWidth/2 - wallThickness/2
  Lwe = -roomWidth/2 + wallThickness/2
  Cwe = roomHeight/2 - wallThickness/2
  Floorwe = -roomHeight/2 + wallThickness/2
  Bwe = -roomDepth/2 + wallThickness/2
  Fwe = roomDepth/2 - wallThickness/2
  if (Xrme >= Rwe or Xlme <= Lwe):
      deltaX = deltaX*(-1)
  if (Ytme >= Cwe or Ybme <= Floorwe):
      deltaY = deltaY*(-1)
  if (Zfme >= Fwe or Zbme <= Bwe):
      deltaZ = deltaZ*(-1)
```

```
 marble.pos = vector(xPos,yPos,zPos)
```



● Run

☑ Big Ball

Choose Ball Speed

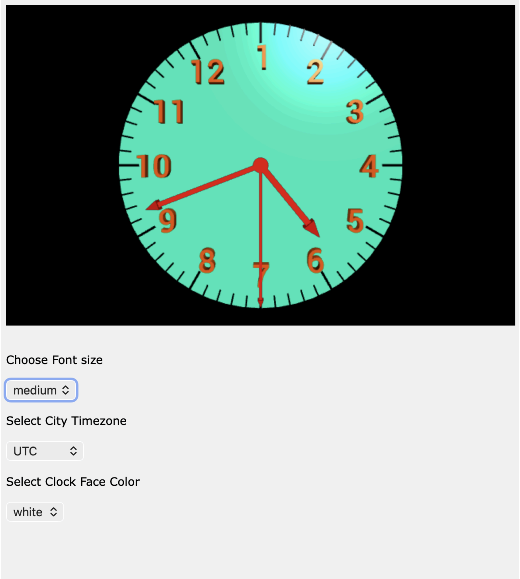
3

Choose Ball Opacity

Python 3D Graphics Tutorial Homework

```
from vpython import *
import numpy as np
import time
clockR = 2
clockT = clockR/10
majorTickL = clockR/7
majorTickT = 2*np.pi*clockR/400
majorTickW = clockT*1.2
manorTickL = clockR/12
manorTickT = 2*np.pi*clockR/600
manorTickW = clockT*1.2
minuteHandL = clockR-majorTickL
minuteHandT = minuteHandL/25
mniuteHandOffset = clockT/2 + minuteHandT
hubRadius = clockT/2
hourHandL = .75*minuteHandL
hourHandT = minuteHandT*1.25
hourHandOffset = clockT/2 + hourHandT
hourRadius = clockT/2
```

```
hourAngle = np.pi/2
minuteAngle = np.pi/2
minInc = .0001
hourInc = minInc/12
secondHandL = clockR - majorTickL/2
secondHandT = minuteHandL/50
secondHandOffset = clockT*1.5 + minuteHandT
secondAngle = np.pi/2
secondInc = minInc*60

cityt = {
  'UTC': 0,
  'New York': -4,
  'London': 0,
  'Berlin': 1,
  'Beijing': 8,
  'Tokyo': 9,
  'Sydney': 10
}
bcolor = {
  'white':color.white,
  'black':color.black,
  'blue':color.blue,
  'cyan':color.cyan,
  'green':color.green
}

currentcity = 'UTC'
currentoffset = cityt[currentcity]

for theta in np.linspace(0,2*np.pi,13):
  majorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),
            color = color.black,length = majorTickL,width = majorTickW,
             height = majorTickT,pos =
vector((clockR-majorTickL/2)*np.cos(theta),(clockR-majorTickL/2)*np.sin(theta),0))
for theta in np.linspace(0,2*np.pi,61):
  manorTick = box(axis = vector(clockR*np.cos(theta),clockR*np.sin(theta),0),
             color = color.black,length = manorTickL,width = manorTickW,height = manorTickT,
             pos =
vector((clockR-manorTickL/2)*np.cos(theta),(clockR-manorTickL/2)*np.sin(theta),0))
clockFace = cylinder(axis = vector(0,0,1),color = vector(0,1,.8),length = clockT,radius =
clockR,pos = vector(0,0,-clockT/2))
minuteHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = minuteHandT,
            length = minuteHandL,pos = vector(0,0,mniuteHandOffset))
hourHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = hourHandT,
           length = hourHandL,pos = vector(0,0,hourHandOffset))
hub = cylinder(axis = vector(0,0,1),color = color.red,radius = hubRadius,length = 2*clockT)
secondHand = arrow(axis = vector(0,1,0),color = color.red,shaftwidth = secondHandT,
            length = secondHandL,pos = vector(0,0,secondHandOffset))
myLabel = label(text=f'{currentcity} Time', pos=vector(0,1.3*clockR,0), height=16, box=False,
color=color.orange)
Angle = np.pi/2
AngleInc = -2*np.pi/12
Angle = Angle + AngleInc
numH = clockR/6

def pickcity(x):
  global currentcity, currentoffset
  currentcity = x.selected
  currentoffset = cityt[currentcity]
  myLabel.text = f'{currentcity} Time'
def backcolor(x):
  clockFace.color = bcolor[x.selected]
scene.append_to_caption('\n\n')
wtext(text = 'Choose Font size')
scene.append_to_caption('\n\n')
def setFontSize(x):
  global numH
  if x.selected == 'small':
     numH = clockR/9
  elif x.selected == 'medium':
     numH = clockR/6
  elif x.selected == 'large':
     numH = clockR/3
  for i, t in enumerate(clockNums):
     theta = np.pi/2 - i*(2*np.pi/12)
     t.height = numH
     t.pos = vector(clockR*.75*np.cos(theta),
                clockR*.75*np.sin(theta) - numH/2, 0)
clockNums = []
menu(bind = setFontSize,choices = ['small','medium','large'])
scene.append_to_caption('\n\n')
wtext(text = "Select City Timezone")
scene.append_to_caption('\n\n')
menu(bind = pickcity, choices = list(cityt.keys()))
scene.append_to_caption('\n\n')
wtext(text = "Select Clock Face Color")
scene.append_to_caption('\n\n')
menu(bind = backcolor, choices = list(bcolor.keys()))
myLabel.text = f'{currentcity} Time'
for i in range(1,13,1):
  clockNum = text(align='center', text=str(i),pos=vector(clockR*.75*np.cos(Angle),
clockR*.75*np.sin(Angle) - numH/2, 0),height=numH, depth=clockT, color=color.orange)
  clockNums.append(clockNum)
  Angle = Angle + AngleInc
while True:
  rate(50)
  utc_ts = time.time()
  citytm = time.gmtime(utc_ts + currentoffset*3600)
  hour   = citytm.tm_hour % 12
  minute = citytm.tm_min
  second = citytm.tm_sec
  hourAngle = -((hour+minute/60)/12)*2*np.pi + np.pi/2
  minuteAngle = -((minute+second/60)/60)*2*np.pi + np.pi/2
  secondAngle = -(second/60)*2*np.pi + np.pi/2
  print(second)
  hourHand.axis = vector(hourHandL*np.cos(hourAngle),hourHandL*np.sin(hourAngle),0)
  minuteHand.axis =
vector(minuteHandL*np.cos(minuteAngle),minuteHandL*np.sin(minuteAngle),0)
  secondHand.axis =
vector(secondHandL*np.cos(secondAngle),secondHandL*np.sin(secondAngle),0)
```



Choose Font size

medium

Select City Timezone

UTC

Select Clock Face Color

white

USA Computing Olympiad and Canadian Computing Competition Practice

```
# ================================
# Hi Mr. Morozov,
# just press Run — a menu will appear.
# If you want to test the problems, simply choose from the menu.
# ================================
# USACO Bronze: Blocked Billboard
# Link: https://usaco.org/index.php?cpid=759&page=viewproblem2
# On the farm, there are two billboards A and B (rectangles aligned
with the axes),
# and a truck T (also a rectangle).
# Each is represented by coordinates (x1, y1, x2, y2), the
bottom-left and top-right points.
# The truck may cover part of the billboards.
# Compute the total visible area of both billboards A and B.
#
# Input format:
# Line 1: ax1 ay1 ax2 ay2 — billboard A
# Line 2: bx1 by1 bx2 by2 — billboard B
# Line 3: tx1 ty1 tx2 ty2 — truck T
#
# Output format:
# A single integer: the total visible area of billboards A and B.
#
# Sample Input:
# 0 0 4 3
# 5 0 8 4
# 2 1 6 3
#
# Sample Output:
# 18
# ================================

def _area(rect):
   x1, y1, x2, y2 = rect
   w = max(0, x2 - x1)
   h = max(0, y2 - y1)
   return w * h

def _overlap(a, b):
   ax1, ay1, ax2, ay2 = a
   bx1, by1, bx2, by2 = b
   w = max(0, min(ax2, bx2) - max(ax1, bx1))
   h = max(0, min(ay2, by2) - max(ay1, by1))
   return w * h

def solve_blocked_billboard():
   print('Enter Test Data To 1.Blocked Billboard')
   ax1, ay1, ax2, ay2 = map(int, input().split())
   bx1, by1, bx2, by2 = map(int, input().split())
```

```
    tx1, ty1, tx2, ty2 = map(int, input().split())

    A = (ax1, ay1, ax2, ay2)
    B = (bx1, by1, bx2, by2)
    T = (tx1, ty1, tx2, ty2)

    visible_A = _area(A) - _overlap(A, T)
    visible_B = _area(B) - _overlap(B, T)

    print(visible_A + visible_B)

# solve_blocked_billboard()

# ================================
# USACO Bronze: Rectangle Pasture
# This problem is not from the official USACO archive.
# It is a simplified practice version created by ChatGPT.
"""
Here are some sample test data:

Input:
3
0 0
3 1
2 5
Expected Output:
15

Test 2
Input:
4
-1 -1
-1 2
3 -1
3 2
Expected Output:
12

Test 3
Input:
2
100 200
```

```
105 210
Expected Output:
50
"""

# Problem description:
# On the farm, there are N cows, each standing at an integer
coordinate (x, y).
# You need to draw an axis-aligned rectangle that contains all the
cows (boundary counts as inside).
# Output the minimum possible area of such a rectangle.
#
# Input format:
# First line: an integer N (1 <= N <= 100)
# Next N lines: two integers xi, yi (-1000 <= xi, yi <= 1000), the
positions of the cows.
#
# Output format:
# One integer: the minimum rectangle area.
#
# Sample Input:
# 3
# 0 0
# 3 1
# 2 5
#
# Sample Output:
# 15
# ================================

def solve1():
   print('Enter Test Data To 2.Rectangle Pasture')
   n = int(input())
   allx = []
   ally = []
   for _ in range(n):
       x, y = map(int, input().split())
       allx.append(x)
       ally.append(y)
   minx = min(allx)
   maxx = max(allx)
   miny = min(ally)
```

```
    maxy = max(ally)
    area = (maxx - minx) * (maxy - miny)
    print(area)

# solve1()

# =================================
# USACO Bronze: Cow Gymnastics
# Link: https://usaco.org/index.php?cpid=963&page=viewproblem2
# Problem description:
# There are K gymnastics practice sessions. Each session lists the
ranking of N cows.
# If cow A is ranked before cow B in all sessions, we say “A is
always better than B.”
# Count how many pairs (A, B) satisfy this condition.
#
# Input format:
# First line: two integers K, N
# Next K lines: each contains N integers, the ranking in one session
(from 1st to Nth).
#
# Output format:
# One integer: the number of pairs (A, B) such that A is always
better than B.
#
# Sample Input:
# 3 4
# 4 1 2 3
# 4 1 3 2
# 4 2 1 3
#
# Sample Output:
# 4
#
# Explanation:
# - Session 1 order: 4 before 1, 1 before 2, 2 before 3
# - Session 2 order: 4 before 1, 1 before 3, 3 before 2
# - Session 3 order: 4 before 2, 2 before 1, 1 before 3
# Valid pairs: (4,1), (4,2), (4,3), (1,3)
# =================================

def always_before(i, j, pos, K):
   for r in range(K):
       if pos[r][i] >= pos[r][j]:
           return False
   return True

def solve2():
   print('Enter Test Data To 3.Cow Gymnastics')
   K, N = map(int, input().split())
   pos = []
   for _ in range(K):
       rank = list(map(int, input().split()))
       one_race_pos = {}
       for idx, cow in enumerate(rank):
           one_race_pos[cow] = idx
       pos.append(one_race_pos)
   ans = 0
   for i in range(1, N+1):
       for j in range(1, N+1):
           if i == j:
               continue
           if always_before(i, j, pos, K):
               ans += 1
   print(ans)

# solve2()

# ================================
# USACO Bronze: Mixing Milk
# Link: https://usaco.org/index.php?cpid=855&page=viewproblem2
# Problem description:
# There are three buckets with capacities c1, c2, c3,
# and initial amounts of milk m1, m2, m3.
# Farmer John performs 100 operations:
#   1st: pour bucket 1 into bucket 2,
#   2nd: pour bucket 2 into bucket 3,
#   3rd: pour bucket 3 into bucket 1,
#   4th: again from bucket 1 into bucket 2 … and so on in a cycle.
#
# Pouring rules:
# - If the target bucket isn’t full, pour as much as possible from
the source.
```

```python
# - If pouring would overflow, only pour until the target is full, leaving some milk in the source.
#
# Task: After 100 operations, output the final amount of milk in each bucket.
#
# Input format:
# c1 m1
# c2 m2
# c3 m3
#
# Output format:
# Three lines:
# amount in bucket 1
# amount in bucket 2
# amount in bucket 3
#
# Sample Input:
# 10 3
# 11 4
# 12 5
#
# Sample Output:
# 0
# 10
# 12
# ================================

def mixing_ops(a, ah, b, bh, c, ch):
    step = 0
    while step < 100:
        if bh + ah <= b:
            bh = bh + ah
            ah = 0
        else:
            ah = bh + ah - b
            bh = b
        step += 1
        if step == 100: break

        if ch + bh <= c:
            ch = ch + bh
```

```
           bh = 0
       else:
           bh = ch + bh - c
           ch = c
       step += 1
       if step == 100: break

       if ah + ch <= a:
           ah = ah + ch
           ch = 0
       else:
           ch = ah + ch - a
           ah = a
       step += 1

   print(ah)
   print(bh)
   print(ch)

def solve3():
   print('Enter Test Data To 4.Mixing Milk')
   a, ah = map(int, input().split())
   b, bh = map(int, input().split())
   c, ch = map(int, input().split())
   mixing_ops(a, ah, b, bh, c, ch)

# solve3()

# ================================
# USACO Bronze: Bucket Brigade
# Link: https://usaco.org/index.php?cpid=939&page=viewproblem2
# Problem description:
# The barn is on fire, and cows want to fetch water from the lake!
# The farm is represented by a 10x10 character grid:
#   - 'B' = Barn (on fire)
#   - 'L' = Lake (source of water)
#   - 'R' = Rock (cannot place cows)
#   - '.' = Empty space (cows can stand here)
#
# Cows must line up in a straight relay to pass water:
# - Water can only move up, down, left, or right
# - A cow must stand adjacent to the lake 'L' to fetch water
# - A cow must stand adjacent to the barn 'B' to put out the fire
# - Cows cannot stand on 'R'
#
# Task: Find the minimum number of cows (on '.' cells) required for the relay.
#
# Input format:
# 10 lines, each with 10 characters ('B', 'L', 'R', '.')
#
# Output format:
# One integer: the minimum number of cows needed.
#
# Sample Input:
# ..........
# ..........
# ..........
# ..B.......
# ..........
# .....R....
# ..........
# ..........
# .....L....
# ..........
#
# Sample Output:
# 7
#
# Explanation:
# - Barn at (4,3), lake at (9,6), rock at (6,6).
# - Shortest distance from lake to barn is 8 steps.
# - Only 7 cows are needed in between.
# ================================

def other(barn, lake):
   maxabsx = max(barn[0], lake[0])
   minabsx = min(barn[0], lake[0])
   maxabsy = max(barn[1], lake[1])
   minabsy = min(barn[1], lake[1])
   an = (maxabsx - minabsx) + (maxabsy - minabsy) - 1
   return an

def lakex_barnx_rockx(barn, lake, rock):
   if barn[0] == lake[0] and lake[0] == rock[0] and max(barn[1],
lake[1]) > rock[1] > min(barn[1], lake[1]):
       maxan = max(barn[1], lake[1])
       minan = min(barn[1], lake[1])
       an = maxan - minan + 1
   else:
       maxan = max(barn[1], lake[1])
       minan = min(barn[1], lake[1])
       an = maxan - minan - 1
   return an

def lakey_barny_rocky(barn, lake, rock):
   if barn[1] == lake[1] and lake[1] == rock[1] and max(barn[0],
lake[0]) > rock[0] > min(barn[0], lake[0]):
       maxan = max(barn[0], lake[0])
       minan = min(barn[0], lake[0])
       an = maxan - minan + 1
   else:
       maxan = max(barn[0], lake[0])
       minan = min(barn[0], lake[0])
       an = maxan - minan - 1
   return an

def solve4():
   print('Enter Test Data To 5.Bucket Brigade')
   grid = [input().strip() for _ in range(10)]
   rock = None
   for y in range(10):
       for x in range(10):
           if grid[y][x] == 'B':
               barn = (x, y)
           if grid[y][x] == 'L':
               lake = (x, y)
           if grid[y][x] == 'R':
               rock = (x, y)

   if rock is not None and barn[0] == lake[0] and lake[0] == rock[0]
and max(barn[1], lake[1]) > rock[1] > min(barn[1], lake[1]):
       print(lakex_barnx_rockx(barn, lake, rock))
   elif rock is not None and barn[1] == lake[1] and lake[1] ==
rock[1] and max(barn[0], lake[0]) > rock[0] > min(barn[0], lake[0]):
       print(lakey_barny_rocky(barn, lake, rock))
```

```
    else:
        print(other(barn, lake))

# solve4()

# ================================
# CCC 2018 S2 – Sunflowers
# Link: https://dmoj.ca/problem/ccc18s2
# Problem background:
# You are given an n × n grid of flower heights.
# The grid may have been rotated clockwise by 0°, 90°, 180°, or 270°.
# Your task is to restore it to the "correct orientation."
#
# Correct orientation definition:
#  - Each row is non-decreasing (left to right).
#  - Each column is non-decreasing (top to bottom).
#
# In other words:
# Looking left-to-right and top-to-bottom, the numbers must not
decrease.
#
# Input format:
# First line: integer n (2 ≤ n ≤ 100), the size of the grid.
# Next n lines: each with n integers, the matrix rows.
#
# Output format:
# Output the rotated matrix (n rows), which is correctly oriented.
#
# Sample Input:
# 3
# 3 7 9
# 2 6 8
# 1 4 5
#
# Sample Output:
# 1 2 3
# 4 6 7
# 5 8 9
#
# Explanation:
# The given matrix was rotated counterclockwise.
# Rotating 90° clockwise produces the correct orientation.
```

```
def mat90(num,mat):
    n = num
    for _ in range(4):
        if check(n,mat):
            return mat
        nmat = []
        for row in zip(*mat[::-1]):
            nmat.append(list(row))
        mat = nmat
    return mat

def check(num,mat):
    n = num
    for i in range(n):
        for x in range(n-1):
            if mat[i][x] > mat[i][x+1]:
                return False
    for i in range(n):
        for x in range(n-1):
            if mat[x][i] > mat[x+1][i]:
                return False
    return True

def solve5():
    print('Enter Test Data To 6.Sunflowers')
    num = int(input().strip())
    mat = [list(map(int, input().split())) for _ in range(num)]
    ans = mat90(num,mat)
    for row in ans:
        print(*row)

#solve5()

# Codeforces 1133C - Balanced Team
# Link: https://codeforces.com/problemset/problem/1133/C
# A school has N students, each with an integer skill level.
# The coach wants to assign as many students as possible into "valid"
teams.
#
# A valid team is defined as:
```

```
#  - Within the same team, the difference between the maximum and
minimum skill
#    levels must be at most 5.
#
# Your Task:
#  - Determine the maximum number of students that can be assigned
into valid teams
#    (maximize the number of students included).
#
# Input Format:
# The first line: an integer N (1 ≤ N ≤ 1000).
# The second line: N integers, representing the students' skill
levels.
#
# Output Format:
# Output a single integer, the maximum number of students that can be
included in teams.
#
# Sample Input:
# 6
# 1 10 17 12 15 2
#
# Sample Output:
# 3
#
# Explanation:
# After sorting the skill levels: [1, 2, 10, 12, 15, 17].
# We can select [10, 12, 15] as one team, since max - min = 15 - 10 =
5.
# Therefore, the maximum number of students in valid teams is 3.

def low_high_nums(num,nums):
    for a in range(num):
        for b in range(num-1):
            if nums[b] > nums[b+1]:
                nums[b],nums[b+1] = nums[b+1],nums[b]
    return nums
def found(num,nums):
    l = 0
    ans = 0
    newnums = low_high_nums(num,nums)
    for i in range(num):
        while newnums[i] - newnums[l] > 5:
                l += 1
        ans = max(ans,i - l + 1)
    return ans

def solve6():
    print('Enter Test Data To 7.Balanced Teams')
    num = int(input().strip())
    nums = list(map(int,input().split()))

    an = found(num,nums)
    print(an)

#solve6()

# ================================
# CCC 2017 J4 - Favourite Times (Practice)
# Link: https://dmoj.ca/problem/ccc17j4
#
# Problem Description:
#
# You have a 12-hour digital clock that shows times from 12:00 up to
11:59.
# Each minute, the time advances by one minute.
#
# A time on the clock is called a "favourite time" if, when the
digits of the
# time are written without the colon, the digits form an arithmetic
sequence.
# That is, the difference between each pair of consecutive digits is
the same.
#
# Examples:
#  - 12:34 → digits 1,2,3,4 → differences are 1,1,1 → arithmetic
sequence ✅
#  - 1:11  → digits 1,1,1 → differences are 0,0 → arithmetic sequence
✅
#  - 2:46  → digits 2,4,6 → differences are 2,2 → arithmetic sequence
✅
#  - 10:08 → digits 1,0,0,8 → differences are -1,0,8 → not arithmetic
❌
#
```

```
# Input format:
#  A single integer N (0 ≤ N ≤ 1,000,000), the number of minutes.
#
# Output format:
#  Output the number of favourite times that will occur in the N
minutes after 12:00.
#
# Explanation:
#  - Start counting from 12:00, after one minute the time is 12:01,
#    after two minutes it is 12:02, etc.
#  - After N minutes, stop.
#  - Count how many of those times were "favourite times".
#
# Sample Input 1:
# 34
# Sample Output 1:
# 1
#
# Sample Input 2:
# 180
# Sample Output 2:
# 11
#
# Sample Input 3:
# 1440
# Sample Output 3:
# 62
# ================================

def solve7():
   print('Enter Test Data To 8.Favourite Time')
   num = int(input().strip())
   hour = 12
   minute = 0
   a = 0
   for i in range(num):
       minute += 1
       if minute > 59:
           minute = 0
           hour += 1
           if hour > 12:
               hour = 1
```

```
        h1 = hour // 10
        h2 = hour % 10
        m1 = minute // 10
        m2 = minute % 10
        if hour < 10:
            clock = [h2, m1, m2]
            if clock[1] - clock[0] == clock[2] - clock[1]:
                a += 1
        else:
            clock = [h1, h2, m1, m2]
            if clock[1] - clock[0] == clock[2] - clock[1] == clock[3] - clock[2]:
                a += 1
    print(a)

#solve7()

# ================================
# CCC 2025 J3 – Product Codes
# Link: https://dmoj.ca/problem/ccc25j3
#
# Problem Description:
#   A store has hired the “Code Cleaning Crew” to update its product codes.
#   Each original product code is a string containing:
#     - uppercase letters (A–Z),
#     - lowercase letters (a–z),
#     - and integers (which may be positive or negative).
#
#   The new product code is formed as follows:
#     1) Remove all lowercase letters.
#     2) Keep all uppercase letters in their original order.
#     3) Find every integer (positive or negative) that appears in the string and sum them.
#     4) Append the resulting sum to the sequence of uppercase letters.
#
# Input format:
#   - The first line contains a positive integer N, the number of product codes.
#   - Each of the next N lines contains one product code string.
#   - It is guaranteed that each string contains at least:
```

```
#       * one uppercase letter,
#       * one lowercase letter,
#       * and one integer (positive or negative).
#   - Sequences of digits that form a number count as a single
integer
#     (e.g., "23" is one integer, not two).
#
# Output format:
#   Output N lines.
#   For each input string, output the transformed product code.
#
# Examples:
#   - "cG23mH-9s" → keep uppercase "GH"; integers are 23 and -9; sum
= 14 → "GH14".
#
# Sample Input 1:
# 1
# AbC3c2Cd9
#
# Sample Output 1:
# ACC14
#
# Sample Input 2:
# 3
# Ahkiy-6ebvXCV1
# 393hhhUHkbs5gh6QpS-9-8
# PL12N-2G1234Duytrty8-86tyaYySsDdEe
#
# Sample Output 2:
# AXCV-5
# UHQS387
# PLNGDYSDE1166
# ================================

def cap(s):
    caps = []
    for ch in s:
        if ch.isupper():
            caps.append(ch)
    caps = ''.join(caps)
    return caps
```

```
def num(s):
    total = 0
    i = 0
    L = len(s)
    while i < L:
        sign = 1
        if s[i] == '-' and i + 1 < L and s[i + 1].isdigit():
            sign = -1
            i += 1
        if i < L and s[i].isdigit():
            val = 0
            while i < L and s[i].isdigit():
                val = val * 10 + int(s[i])
                i += 1
            total += sign * val
        else:
            i += 1
    return total

def solve8():
    print('Enter Test Data To 9.product codes')
    n = int(input().strip())
    s = [input().strip() for _ in range(n)]
    print('')
    print('Output')
    for ch in s:
        caps = cap(ch)
        nums = num(ch)
        print(f"{caps}{nums}")

#solve8()

# ================================
# 🇨🇦 CCC 2019 J2 - Time to Decompress
# Link: https://dmoj.ca/problem/ccc19j2
# ================================
#
# Problem Description:
# You will be given a sequence of lines.
# Each line will contain a positive integer, followed by a single space,
# followed by a character (either a letter or a punctuation mark).
# You must output the character repeated the specified number of
times.
#
# Input Specification:
# The first line of input contains an integer L (1 ≤ L ≤ 5),
# representing the number of lines that follow.
#
# Each of the next L lines contains an integer N (1 ≤ N ≤ 80)
# and a character C.
#
# Output Specification:
# For each of the L input lines,
# output a line containing the character C repeated N times.
#
# Sample Input:
# 4
# 9 +
# 3 -
# 12 A
# 2 X
#
# Sample Output:
# +++++++++
# ---
# AAAAAAAAAAAA
# XX
# ================================

def solve9():
   print('Enter Test Data To 10.Time to Decompress')
   num = int(input().strip())
   data = [input().split() for _ in range(num)]
   for n, s in data:
       n = int(n)
       print(s * n)

#solve9()

# ================================
# CCC 2021 J1 - Boiling Water
# Link: https://dmoj.ca/problem/ccc21j1
# ================================
#
# Problem Description:
#
# When water is heated, it boils at a certain temperature.
# A scientist wants to know how the atmospheric pressure changes
# as the temperature changes.
#
# The relationship between the atmospheric pressure P and
# the temperature B (in degrees Celsius) is given by the formula:
#
#       P = 5 × B − 400
#
# Your task is to:
#   1. Read an integer B representing the temperature (in °C).
#   2. Calculate and output the value of P.
#   3. Output one more line indicating whether the pressure is:
#        • above sea level (if P > 100, output 1)
#        • at sea level (if P == 100, output 0)
#        • below sea level (if P < 100, output -1)
#
# ------------------------
# Input Specification:
# The input will contain one integer B (0 ≤ B ≤ 1000).
#
# ------------------------
# Output Specification:
# Output two lines:
#   Line 1: the calculated pressure P
#   Line 2: one of the integers 1, 0, or -1
#
# ------------------------
# Sample Input 1:
# 80
#
# Sample Output 1:
# 0
# -1
#
# ------------------------
# Sample Input 2:
# 150
#
```

```python
# Sample Output 2:
# 350
# 1
#
# ================================

def solve10():
    print('Enter Test Data To 11.Boiling Water')
    b = int(input())
    p = 5 * b - 400
    print(p)
    if p > 100:
        print(1)
    elif p == 100:
        print(0)
    else:
        print(-1)

#solve10()

# ================================
# USACO Bronze: Shell Game
# Source: USACO 2019 January Contest, Bronze
# Link: https://usaco.org/index.php?page=viewproblem2&cpid=891
# ================================
#
# Farmer John is playing a shell game with Bessie the cow.
# He places three shells on a table, labeled with the numbers 1, 2, and 3.
# He then places a pebble under one of these shells.
#
# Farmer John then performs N moves.
# Each move consists of two parts:
#   1. He swaps the shells at two given positions a and b.
#   2. Bessie guesses which shell currently contains the pebble (she guesses shell g).
#
# Your task is to determine the maximum number of correct guesses
# Bessie could have made if she had initially known which shell the pebble was under.
#
# That is, since we don't know where the pebble started,
# you must consider all three possible initial positions (1, 2, 3)
# and determine the maximum number of times Bessie could have guessed
correctly.
#
# ----------------
# Input Format:
# ----------------
# Line 1: The integer N (1 ≤ N ≤ 100) — the number of moves.
# Lines 2..N+1: Each line contains three integers a, b, g.
#   - a and b are the two shell positions being swapped.
#   - g is Bessie's guess (the position she thinks the pebble is
under).
#
# ----------------
# Output Format:
# ----------------
# A single integer — the maximum number of correct guesses Bessie
could have made.
#
# ----------------
# Sample Input:
# ----------------
# 3
# 1 2 1
# 3 2 1
# 1 3 1
#
# ----------------
# Sample Output:
# ----------------
# 2
#
# ----------------
# Explanation:
# ----------------
# If the pebble started under shell 1 -> 1 correct guess.
# If the pebble started under shell 2 -> 2 correct guesses.
# If the pebble started under shell 3 -> 1 correct guess.
# Therefore, the maximum possible number of correct guesses is 2.
# ================================

def solve11():
```

```python
    print('Enter Test Data To 12.Shell Game')
    num = int(input())
    ops = [list(map(int, input().split())) for _ in range(num)]
    an = 0
    for start in [1,2,3]:
        pearl = start
        correct = 0
        for a,b,g in ops:
            if pearl == a:
                pearl = b
            elif pearl == b:
                pearl = a
            if pearl == g:
                correct += 1
        an = max(correct,an)
    print(an)

#solve11()

# ================================================
# 🐮 USACO 2024 January Contest, Bronze Division
# Problem: Cow College
# Link: https://usaco.org/index.php?page=viewproblem2&cpid=1377
# ================================================

# Farmer John has just opened a new school for his cows called "Cow College"!
#
# He has surveyed N cows to determine how much each one would be willing to pay for tuition.
# The i-th cow is willing to pay at most Pi dollars.
#
# Farmer John must choose a single tuition price T (an integer).
# Every cow who is willing to pay at least T (that is, Pi ≥ T) will enroll in the college.
#
# The total revenue is then:
#     revenue = T × (number of cows whose Pi ≥ T)
#
# Farmer John wants to choose the tuition price T that maximizes his total revenue.
# If there are multiple prices that yield the same maximum revenue,
# he should choose the **smallest such T**.
#
# -----------------------------------------------
# INPUT FORMAT (from standard input):
# Line 1: The integer N (1 ≤ N ≤ 100,000)
# Next N lines: Each line contains one integer Pi (1 ≤ Pi ≤ 1,000,000,000)
#
# -----------------------------------------------
# OUTPUT FORMAT (to standard output):
# Print two integers separated by a space:
#     1️⃣ The optimal tuition price T
#     2️⃣ The maximum possible total revenue
#
# -----------------------------------------------
# SAMPLE INPUT:
# 4
# 2
# 8
# 10
# 7
#
# SAMPLE OUTPUT:
# 7 21
#
# EXPLANATION:
# If T = 2 → all 4 cows enroll → revenue = 2 × 4 = 8
# If T = 7 → cows paying [7, 8, 10] enroll → 3 × 7 = 21 ✅
# If T = 8 → cows paying [8, 10] enroll → 2 × 8 = 16
# If T = 10 → only one cow enrolls → 1 × 10 = 10
# The best choice is T = 7 with revenue = 21.
# ================================================

def solve12():
    print('Enter Test Data To 13.Cow College')
    num = int(input())
    nums = [int(input()) for _ in range(num)]
    result = 0
    an = 0
    ans = 0
    for i in range(num):
        bigger = 0
        for j in range(num):
            if nums[j] >= nums[i]:
                bigger += 1
        newresult = nums[i] * bigger
        if newresult > result:
            result = newresult
            an = nums[i]
            ans = newresult
    print(an,ans)

#solve12()

# ============================================
# CCC 2020 Senior 2: Escape Room
# Link:
https://cemc.uwaterloo.ca/contests/computing/2020/stage%201/seniorEn.
pdf
# ============================================

# Problem Description:
# You are given an R by C grid of positive integers.
# You start in the top-left corner (1, 1) and want to reach the
bottom-right corner (R, C).
#
# You can move from a cell with integer value v to any other cell (r,
c)
# such that r × c = v.
#
# For example, if the cell contains the number 6,
# then you can move to cells (1, 6), (2, 3), (3, 2), or (6, 1),
# as long as those cells exist within the boundaries of the grid.
#
# Your task is to determine whether it is possible
# to reach the bottom-right corner (R, C) starting from the top-left
corner (1, 1).

# --------------------------------------------
# Input Specification:
# The first line contains two integers R and C (1 ≤ R, C ≤ 1000).
# The next R lines each contain C positive integers,
# representing the values in each cell of the grid.
#
```

```python
# ---------------------------------------------
# Output Specification:
# Output “yes” if it is possible to reach (R, C).
# Otherwise, output “no”.
#
# ---------------------------------------------
# Sample Input 1:
# 3 4
# 3 10 8 14
# 1 11 12 12
# 6 2 3 9
#
# Sample Output 1:
# yes
#
# Explanation:
# One possible sequence of moves is:
# (1,1) → (3,1) → (3,2) → (2,3) → (1,3) → (1,4) → (3,4)
# which reaches the bottom-right corner.
#
# ---------------------------------------------
# Sample Input 2:
# 2 2
# 2 4
# 6 9
#
# Sample Output 2:
# no
# =============================================

def solve13():
    R, C = map(int, input().split())
    grid = [list(map(int, input().split())) for _ in range(R)]

    stack = [(1, 1)]
    visited = set([(1, 1)])

    while stack:
        r, c = stack.pop()
        if (r, c) == (R, C):
            print("yes")
            return
        v = grid[r - 1][c - 1]

        i = 1
        while i * i <= v:
            if v % i == 0:
                r1, c1 = i, v // i
                if 1 <= r1 <= R and 1 <= c1 <= C and (r1, c1) not in visited:
                    visited.add((r1, c1))
                    stack.append((r1, c1))

                r2, c2 = v // i, i
                if (r2, c2) != (r1, c1):
                    if 1 <= r2 <= R and 1 <= c2 <= C and (r2, c2) not in visited:
                        visited.add((r2, c2))
                        stack.append((r2, c2))
            i += 1

    print("no")

#solve13()

# ================================================================
# CCC 2016 S2: Tandem Bicycle
# Link: https://dmoj.ca/problem/ccc16s2
# ================================================================
#
# Problem Description:
#
# Farmers and city riders are competing in a tandem bicycle race.
# Each tandem bicycle is ridden by two riders: one from the farm and one from the city.
#
# The speed of a tandem bicycle is equal to the **maximum speed** of its two riders.
#
# You are given the speed of each rider in both groups, and an integer `type`:
#   - If `type = 1`, you must minimize the total speed of all tandem bicycles.
#   - If `type = 2`, you must maximize the total speed of all tandem
bicycles.
#
# Each rider must be used exactly once.
#
# ---------------------------------------------------------------
# Input Specification:
# The first line contains an integer `type` (1 or 2).
# The second line contains an integer `n` (1 ≤ n ≤ 1000),
#   the number of riders in each group.
# The third line contains `n` space-separated integers,
#   the speeds of the first group of riders.
# The fourth line contains `n` space-separated integers,
#   the speeds of the second group of riders.
#
# ---------------------------------------------------------------
# Output Specification:
# Output one integer — the minimum or maximum possible total speed
# of all tandem bicycles, depending on the value of `type`.
#
# ---------------------------------------------------------------
# Sample Input 1:
# 1
# 3
# 5 1 4
# 6 2 4
#
# Sample Output 1:
# 12
#
# ---------------------------------------------------------------
# Sample Input 2:
# 2
# 3
# 5 1 4
# 6 2 4
#
# Sample Output 2:
# 15
#
# ---------------------------------------------------------------
# Notes:
```

```python
# - For `type = 1`, both lists should be sorted in increasing order
#   to minimize the total.
# - For `type = 2`, one list should be sorted in increasing order
#   and the other in decreasing order to maximize the total.
#
# ==============================================================

def low_high(num,nums):
   for i in range(num):
       for x in range(i + 1, num):
           if nums[i] > nums[x]:
               nums[i], nums[x] = nums[x], nums[i]
   return nums

def high_low(num,nums):
   for i in range(num):
       for x in range(i + 1,num):
           if nums[i] < nums[x]:
               nums[i], nums[x] = nums[x], nums[i]
   return nums

def solve14():
   type = int(input())
   num = int(input())
   farmers = list(map(int,input().split()))
   city = list(map(int,input().split()))

   if type == 1:
       farmers = high_low(num,farmers)
       city = high_low(num,city)
       an = 0
       for i in range(num):
           an += max(farmers[i],city[i])
       print(an)

   elif type == 2:
       farmers = high_low(num,farmers)
       city = low_high(num,city)
       an = 0
       for i in range(num):
           an += max(farmers[i],city[i])
       print(an)
```

```
#solve14()

while True:
   print('')
   print('Test: 1.Blocked Billboard | 2.Rectangle Pasture | 3.Cow
Gymnastics')
   print('4.Mixing Milk | 5.Bucket Brigade | 6.Sunflowers |
7.Balanced Teams')
   print('8.Favourite Times | 9.Product Codes | 10.Time to Decompress
')
   print('11.Boiling Water | 12.Shell Game | 13.Cow College |
14.Escape Room')
   print('15.Tandem Bicycle | Exit')
   print('-'*66)
   an = input('>>>').strip()
   if an == "1" or an.lower() == "blocked billboard" or an.lower() ==
"1.blocked billboard":
       solve_blocked_billboard()
   elif an == "2" or an.lower() == "rectangle pasture" or an.lower()
== "2.rectangle pasture":
       solve1()
   elif an == "3" or an.lower() == "cow gymnastics" or an.lower() ==
"3.cow gymnastics":
       solve2()
   elif an == "4" or an.lower() == "mixing milk" or an.lower() ==
"4.mixing milk":
       solve3()
   elif an == "5" or an.lower() == "bucket brigade" or an.lower() ==
"5.bucket brigade":
       solve4()
   elif an == "6" or an.lower() == "sunflowers" or an.lower() ==
"6.sunflowers":
       solve5()
   elif an == "7" or an.lower() == "balanced teams" or an.lower() ==
"7.balanced teams":
       solve6()
   elif an == "8" or an.lower() == "favourite times" or an.lower() ==
"8.favourite times":
       solve7()
   elif an == "9" or an.lower() == "product codes" or an.lower() ==
"9.product codes":
       solve8()
   elif an == "10" or an.lower() == 'time to decompress' or
an.lower() == "10.time to decompress":
       solve9()
   elif an == "11" or an .lower() == 'boiling water' or an.lower() ==
"11.boiling water":
       solve10()
   elif an == '12' or an.lower() == 'shell game' or an.lower() ==
'12.shell game':
       solve11()
   elif an == '13' or an.lower() == 'cow college' or an.lower() ==
'13.cow college':
       solve12()
   elif an == '14' or an.lower() == 'escape room' or an.lower() ==
'14.escape room':
       solve13()
   elif an == '15' or an.lower() == 'tandem bicycle' or an.lower() ==
'15.tandem bicycle':
       solve14()
   elif an.lower() == "exit":
       break
# ================================
# Hi Mr. Morozov,
# just press Run — a menu will appear.
# If you want to test the problems, simply choose from the menu.
# ================================
# USACO Bronze: Blocked Billboard
# Link: https://usaco.org/index.php?cpid=759&page=viewproblem2
# On the farm, there are two billboards A and B (rectangles aligned
with the axes),
# and a truck T (also a rectangle).
# Each is represented by coordinates (x1, y1, x2, y2), the
bottom-left and top-right points.
# The truck may cover part of the billboards.
# Compute the total visible area of both billboards A and B.
#
# Input format:
# Line 1: ax1 ay1 ax2 ay2 —— billboard A
# Line 2: bx1 by1 bx2 by2 —— billboard B
# Line 3: tx1 ty1 tx2 ty2 —— truck T
#
# Output format:
```

```
# A single integer: the total visible area of billboards A and B.
#
# Sample Input:
# 0 0 4 3
# 5 0 8 4
# 2 1 6 3
#
# Sample Output:
# 18
# ================================

def _area(rect):
   x1, y1, x2, y2 = rect
   w = max(0, x2 - x1)
   h = max(0, y2 - y1)
   return w * h

def _overlap(a, b):
   ax1, ay1, ax2, ay2 = a
   bx1, by1, bx2, by2 = b
   w = max(0, min(ax2, bx2) - max(ax1, bx1))
   h = max(0, min(ay2, by2) - max(ay1, by1))
   return w * h

def solve_blocked_billboard():
   print('Enter Test Data To 1.Blocked Billboard')
   ax1, ay1, ax2, ay2 = map(int, input().split())
   bx1, by1, bx2, by2 = map(int, input().split())
   tx1, ty1, tx2, ty2 = map(int, input().split())

   A = (ax1, ay1, ax2, ay2)
   B = (bx1, by1, bx2, by2)
   T = (tx1, ty1, tx2, ty2)

   visible_A = _area(A) - _overlap(A, T)
   visible_B = _area(B) - _overlap(B, T)

   print(visible_A + visible_B)

# solve_blocked_billboard()

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

```python
# USACO Bronze: Rectangle Pasture
# This problem is not from the official USACO archive.
# It is a simplified practice version created by ChatGPT.
"""
Here are some sample test data:

Input:
3
0 0
3 1
2 5
Expected Output:
15

Test 2
Input:
4
-1 -1
-1 2
3 -1
3 2
Expected Output:
12

Test 3
Input:
2
100 200
105 210
Expected Output:
50
"""

# Problem description:
# On the farm, there are N cows, each standing at an integer coordinate (x, y).
# You need to draw an axis-aligned rectangle that contains all the cows (boundary counts as inside).
# Output the minimum possible area of such a rectangle.
#
# Input format:
# First line: an integer N (1 <= N <= 100)
# Next N lines: two integers xi, yi (-1000 <= xi, yi <= 1000), the positions of the cows.
#
# Output format:
# One integer: the minimum rectangle area.
#
# Sample Input:
# 3
# 0 0
# 3 1
# 2 5
#
# Sample Output:
# 15
# ================================

def solve1():
   print('Enter Test Data To 2.Rectangle Pasture')
   n = int(input())
   allx = []
   ally = []
   for _ in range(n):
       x, y = map(int, input().split())
       allx.append(x)
       ally.append(y)
   minx = min(allx)
   maxx = max(allx)
   miny = min(ally)
   maxy = max(ally)
   area = (maxx - minx) * (maxy - miny)
   print(area)

# solve1()

# ================================
# USACO Bronze: Cow Gymnastics
# Link: https://usaco.org/index.php?cpid=963&page=viewproblem2
# Problem description:
# There are K gymnastics practice sessions. Each session lists the ranking of N cows.
# If cow A is ranked before cow B in all sessions, we say “A is always better than B.”
# Count how many pairs (A, B) satisfy this condition.
#
# Input format:
# First line: two integers K, N
# Next K lines: each contains N integers, the ranking in one session
(from 1st to Nth).
#
# Output format:
# One integer: the number of pairs (A, B) such that A is always
better than B.
#
# Sample Input:
# 3 4
# 4 1 2 3
# 4 1 3 2
# 4 2 1 3
#
# Sample Output:
# 4
#
# Explanation:
# - Session 1 order: 4 before 1, 1 before 2, 2 before 3
# - Session 2 order: 4 before 1, 1 before 3, 3 before 2
# - Session 3 order: 4 before 2, 2 before 1, 1 before 3
# Valid pairs: (4,1), (4,2), (4,3), (1,3)
# ================================

def always_before(i, j, pos, K):
    for r in range(K):
        if pos[r][i] >= pos[r][j]:
            return False
    return True

def solve2():
    print('Enter Test Data To 3.Cow Gymnastics')
    K, N = map(int, input().split())
    pos = []
    for _ in range(K):
        rank = list(map(int, input().split()))
        one_race_pos = {}
        for idx, cow in enumerate(rank):
            one_race_pos[cow] = idx
        pos.append(one_race_pos)
    ans = 0
    for i in range(1, N+1):
        for j in range(1, N+1):
            if i == j:
                continue
            if always_before(i, j, pos, K):
                ans += 1
    print(ans)

# solve2()

# ================================
# USACO Bronze: Mixing Milk
# Link: https://usaco.org/index.php?cpid=855&page=viewproblem2
# Problem description:
# There are three buckets with capacities c1, c2, c3,
# and initial amounts of milk m1, m2, m3.
# Farmer John performs 100 operations:
#   1st: pour bucket 1 into bucket 2,
#   2nd: pour bucket 2 into bucket 3,
#   3rd: pour bucket 3 into bucket 1,
#   4th: again from bucket 1 into bucket 2 … and so on in a cycle.
#
# Pouring rules:
# - If the target bucket isn’t full, pour as much as possible from the source.
# - If pouring would overflow, only pour until the target is full, leaving some milk in the source.
#
# Task: After 100 operations, output the final amount of milk in each bucket.
#
# Input format:
# c1 m1
# c2 m2
# c3 m3
#
# Output format:
# Three lines:
# amount in bucket 1
# amount in bucket 2
# amount in bucket 3
#
# Sample Input:
# 10 3
# 11 4
# 12 5
#
# Sample Output:
# 0
# 10
# 12
# ================================

def mixing_ops(a, ah, b, bh, c, ch):
   step = 0
   while step < 100:
       if bh + ah <= b:
           bh = bh + ah
           ah = 0
       else:
           ah = bh + ah - b
           bh = b
       step += 1
       if step == 100: break

       if ch + bh <= c:
           ch = ch + bh
           bh = 0
       else:
           bh = ch + bh - c
           ch = c
       step += 1
       if step == 100: break

       if ah + ch <= a:
           ah = ah + ch
           ch = 0
       else:
           ch = ah + ch - a
           ah = a
       step += 1
   print(ah)
   print(bh)
   print(ch)

def solve3():
   print('Enter Test Data To 4.Mixing Milk')
   a, ah = map(int, input().split())
   b, bh = map(int, input().split())
   c, ch = map(int, input().split())
   mixing_ops(a, ah, b, bh, c, ch)

# solve3()

# ================================
# USACO Bronze: Bucket Brigade
# Link: https://usaco.org/index.php?cpid=939&page=viewproblem2
# Problem description:
# The barn is on fire, and cows want to fetch water from the lake!
# The farm is represented by a 10x10 character grid:
#   - 'B' = Barn (on fire)
#   - 'L' = Lake (source of water)
#   - 'R' = Rock (cannot place cows)
#   - '.' = Empty space (cows can stand here)
#
# Cows must line up in a straight relay to pass water:
# - Water can only move up, down, left, or right
# - A cow must stand adjacent to the lake 'L' to fetch water
# - A cow must stand adjacent to the barn 'B' to put out the fire
# - Cows cannot stand on 'R'
#
# Task: Find the minimum number of cows (on '.' cells) required for
the relay.
#
# Input format:
# 10 lines, each with 10 characters ('B', 'L', 'R', '.')
#
# Output format:
# One integer: the minimum number of cows needed.
#
# Sample Input:
# ..........
# ..........
# ..........
# ..B.......
# ..........
# .....R....
# ..........
# ..........
# .....L....
# ..........
#
# Sample Output:
# 7
#
# Explanation:
# - Barn at (4,3), lake at (9,6), rock at (6,6).
# - Shortest distance from lake to barn is 8 steps.
# - Only 7 cows are needed in between.
# ================================

def other(barn, lake):
   maxabsx = max(barn[0], lake[0])
   minabsx = min(barn[0], lake[0])
   maxabsy = max(barn[1], lake[1])
   minabsy = min(barn[1], lake[1])
   an = (maxabsx - minabsx) + (maxabsy - minabsy) - 1
   return an

def lakex_barnx_rockx(barn, lake, rock):
   if barn[0] == lake[0] and lake[0] == rock[0] and max(barn[1],
lake[1]) > rock[1] > min(barn[1], lake[1]):
       maxan = max(barn[1], lake[1])
       minan = min(barn[1], lake[1])
       an = maxan - minan + 1
   else:
       maxan = max(barn[1], lake[1])
       minan = min(barn[1], lake[1])
       an = maxan - minan - 1
   return an

def lakey_barny_rocky(barn, lake, rock):
   if barn[1] == lake[1] and lake[1] == rock[1] and max(barn[0],
lake[0]) > rock[0] > min(barn[0], lake[0]):
       maxan = max(barn[0], lake[0])
       minan = min(barn[0], lake[0])
       an = maxan - minan + 1
   else:
       maxan = max(barn[0], lake[0])
       minan = min(barn[0], lake[0])
       an = maxan - minan - 1
   return an

def solve4():
   print('Enter Test Data To 5.Bucket Brigade')
   grid = [input().strip() for _ in range(10)]
   rock = None
   for y in range(10):
       for x in range(10):
           if grid[y][x] == 'B':
               barn = (x, y)
           if grid[y][x] == 'L':
               lake = (x, y)
           if grid[y][x] == 'R':
               rock = (x, y)

   if rock is not None and barn[0] == lake[0] and lake[0] == rock[0]
and max(barn[1], lake[1]) > rock[1] > min(barn[1], lake[1]):
       print(lakex_barnx_rockx(barn, lake, rock))
   elif rock is not None and barn[1] == lake[1] and lake[1] ==
rock[1] and max(barn[0], lake[0]) > rock[0] > min(barn[0], lake[0]):
       print(lakey_barny_rocky(barn, lake, rock))
   else:
       print(other(barn, lake))

# solve4()

# ================================
# CCC 2018 S2 - Sunflowers
# Link: https://dmoj.ca/problem/ccc18s2
# Problem background:
# You are given an n × n grid of flower heights.
# The grid may have been rotated clockwise by 0°, 90°, 180°, or 270°.
# Your task is to restore it to the "correct orientation."
#
# Correct orientation definition:
#  - Each row is non-decreasing (left to right).
#  - Each column is non-decreasing (top to bottom).
#
# In other words:
# Looking left-to-right and top-to-bottom, the numbers must not
decrease.
#
# Input format:
# First line: integer n (2 ≤ n ≤ 100), the size of the grid.
# Next n lines: each with n integers, the matrix rows.
#
# Output format:
# Output the rotated matrix (n rows), which is correctly oriented.
#
# Sample Input:
# 3
# 3 7 9
# 2 6 8
# 1 4 5
#
# Sample Output:
# 1 2 3
# 4 6 7
# 5 8 9
#
# Explanation:
# The given matrix was rotated counterclockwise.
# Rotating 90° clockwise produces the correct orientation.

def mat90(num,mat):
    n = num
    for _ in range(4):
        if check(n,mat):
            return mat
        nmat = []
        for row in zip(*mat[::-1]):
            nmat.append(list(row))
        mat = nmat
    return mat

def check(num,mat):
    n = num
    for i in range(n):
        for x in range(n-1):
            if mat[i][x] > mat[i][x+1]:
                return False
    for i in range(n):
        for x in range(n-1):
            if mat[x][i] > mat[x+1][i]:
                return False
    return True

def solve5():
    print('Enter Test Data To 6.Sunflowers')
    num = int(input().strip())
    mat = [list(map(int, input().split())) for _ in range(num)]
    ans = mat90(num,mat)
    for row in ans:
        print(*row)

#solve5()

# Codeforces 1133C - Balanced Team
# Link: https://codeforces.com/problemset/problem/1133/C
# A school has N students, each with an integer skill level.
# The coach wants to assign as many students as possible into "valid"
teams.
#
# A valid team is defined as:
#  - Within the same team, the difference between the maximum and
minimum skill
#    levels must be at most 5.
#
# Your Task:
#  - Determine the maximum number of students that can be assigned
into valid teams
#    (maximize the number of students included).
#
# Input Format:
# The first line: an integer N (1 ≤ N ≤ 1000).
# The second line: N integers, representing the students' skill
levels.
#
# Output Format:
```

```python
# Output a single integer, the maximum number of students that can be included in teams.
#
# Sample Input:
# 6
# 1 10 17 12 15 2
#
# Sample Output:
# 3
#
# Explanation:
# After sorting the skill levels: [1, 2, 10, 12, 15, 17].
# We can select [10, 12, 15] as one team, since max - min = 15 - 10 = 5.
# Therefore, the maximum number of students in valid teams is 3.

def low_high_nums(num,nums):
    for a in range(num):
        for b in range(num-1):
            if nums[b] > nums[b+1]:
                nums[b],nums[b+1] = nums[b+1],nums[b]
    return nums
def found(num,nums):
    l = 0
    ans = 0
    newnums = low_high_nums(num,nums)
    for i in range(num):
        while newnums[i] - newnums[l] > 5:
                l += 1
        ans = max(ans,i - l + 1)
    return ans

def solve6():
    print('Enter Test Data To 7.Balanced Teams')
    num = int(input().strip())
    nums = list(map(int,input().split()))

    an = found(num,nums)
    print(an)

#solve6()
```

```
# =================================
# CCC 2017 J4 – Favourite Times (Practice)
# Link: https://dmoj.ca/problem/ccc17j4
#
# Problem Description:
#
# You have a 12-hour digital clock that shows times from 12:00 up to 11:59.
# Each minute, the time advances by one minute.
#
# A time on the clock is called a "favourite time" if, when the digits of the
# time are written without the colon, the digits form an arithmetic sequence.
# That is, the difference between each pair of consecutive digits is the same.
#
# Examples:
#  - 12:34 → digits 1,2,3,4 → differences are 1,1,1 → arithmetic sequence ✅
#  - 1:11  → digits 1,1,1 → differences are 0,0 → arithmetic sequence ✅
#  - 2:46  → digits 2,4,6 → differences are 2,2 → arithmetic sequence ✅
#  - 10:08 → digits 1,0,0,8 → differences are -1,0,8 → not arithmetic ❌
#
# Input format:
#  A single integer N (0 ≤ N ≤ 1,000,000), the number of minutes.
#
# Output format:
#  Output the number of favourite times that will occur in the N minutes after 12:00.
#
# Explanation:
#  - Start counting from 12:00, after one minute the time is 12:01,
#    after two minutes it is 12:02, etc.
#  - After N minutes, stop.
#  - Count how many of those times were "favourite times".
#
# Sample Input 1:
# 34
# Sample Output 1:
# 1
#
# Sample Input 2:
# 180
# Sample Output 2:
# 11
#
# Sample Input 3:
# 1440
# Sample Output 3:
# 62
# ================================

def solve7():
   print('Enter Test Data To 8.Favourite Time')
   num = int(input().strip())
   hour = 12
   minute = 0
   a = 0
   for i in range(num):
       minute += 1
       if minute > 59:
           minute = 0
           hour += 1
           if hour > 12:
               hour = 1
       h1 = hour // 10
       h2 = hour % 10
       m1 = minute // 10
       m2 = minute % 10
       if hour < 10:
           clock = [h2, m1, m2]
           if clock[1] - clock[0] == clock[2] - clock[1]:
               a += 1
       else:
           clock = [h1, h2, m1, m2]
           if clock[1] - clock[0] == clock[2] - clock[1] == clock[3]
- clock[2]:
               a += 1
   print(a)
```

```
#solve7()

# =================================
# CCC 2025 J3 - Product Codes
# Link: https://dmoj.ca/problem/ccc25j3
#
# Problem Description:
#   A store has hired the “Code Cleaning Crew” to update its product codes.
#   Each original product code is a string containing:
#     - uppercase letters (A-Z),
#     - lowercase letters (a-z),
#     - and integers (which may be positive or negative).
#
#   The new product code is formed as follows:
#     1) Remove all lowercase letters.
#     2) Keep all uppercase letters in their original order.
#     3) Find every integer (positive or negative) that appears in the string and sum them.
#     4) Append the resulting sum to the sequence of uppercase letters.
#
# Input format:
#   - The first line contains a positive integer N, the number of product codes.
#   - Each of the next N lines contains one product code string.
#   - It is guaranteed that each string contains at least:
#       * one uppercase letter,
#       * one lowercase letter,
#       * and one integer (positive or negative).
#   - Sequences of digits that form a number count as a single integer
#     (e.g., "23" is one integer, not two).
#
# Output format:
#   Output N lines.
#   For each input string, output the transformed product code.
#
# Examples:
#   - "cG23mH-9s" → keep uppercase "GH"; integers are 23 and -9; sum = 14 → "GH14".
#
```

```python
# Sample Input 1:
# 1
# AbC3c2Cd9
#
# Sample Output 1:
# ACC14
#
# Sample Input 2:
# 3
# Ahkiy-6ebvXCV1
# 393hhhUHkbs5gh6QpS-9-8
# PL12N-2G1234Duytrty8-86tyaYySsDdEe
#
# Sample Output 2:
# AXCV-5
# UHQS387
# PLNGDYSDE1166
# ================================

def cap(s):
    caps = []
    for ch in s:
        if ch.isupper():
            caps.append(ch)
    caps = ''.join(caps)
    return caps

def num(s):
    total = 0
    i = 0
    L = len(s)
    while i < L:
        sign = 1
        if s[i] == '-' and i + 1 < L and s[i + 1].isdigit():
            sign = -1
            i += 1
        if i < L and s[i].isdigit():
            val = 0
            while i < L and s[i].isdigit():
                val = val * 10 + int(s[i])
                i += 1
            total += sign * val
        else:
            i += 1
    return total

def solve8():
    print('Enter Test Data To 9.product codes')
    n = int(input().strip())
    s = [input().strip() for _ in range(n)]
    print('')
    print('Output')
    for ch in s:
        caps = cap(ch)
        nums = num(ch)
        print(f"{caps}{nums}")

#solve8()

# ================================
# 🇨🇦 CCC 2019 J2 - Time to Decompress
# Link: https://dmoj.ca/problem/ccc19j2
# ================================
#
# Problem Description:
# You will be given a sequence of lines.
# Each line will contain a positive integer, followed by a single space,
# followed by a character (either a letter or a punctuation mark).
# You must output the character repeated the specified number of times.
#
# Input Specification:
# The first line of input contains an integer L (1 ≤ L ≤ 5),
# representing the number of lines that follow.
#
# Each of the next L lines contains an integer N (1 ≤ N ≤ 80)
# and a character C.
#
# Output Specification:
# For each of the L input lines,
# output a line containing the character C repeated N times.
#
# Sample Input:
```

```
# 4
# 9 +
# 3 -
# 12 A
# 2 X
#
# Sample Output:
# +++++++++
# ---
# AAAAAAAAAAAA
# XX
# ================================

def solve9():
   print('Enter Test Data To 10.Time to Decompress')
   num = int(input().strip())
   data = [input().split() for _ in range(num)]
   for n, s in data:
       n = int(n)
       print(s * n)

#solve9()

# ================================
# CCC 2021 J1 - Boiling Water
# Link: https://dmoj.ca/problem/ccc21j1
# ================================
#
# Problem Description:
#
# When water is heated, it boils at a certain temperature.
# A scientist wants to know how the atmospheric pressure changes
# as the temperature changes.
#
# The relationship between the atmospheric pressure P and
# the temperature B (in degrees Celsius) is given by the formula:
#
#       P = 5 × B - 400
#
# Your task is to:
#   1. Read an integer B representing the temperature (in °C).
#   2. Calculate and output the value of P.
```

```python
#   3. Output one more line indicating whether the pressure is:
#        • above sea level (if P > 100, output 1)
#        • at sea level (if P == 100, output 0)
#        • below sea level (if P < 100, output -1)
#
# ------------------------
# Input Specification:
# The input will contain one integer B (0 ≤ B ≤ 1000).
#
# ------------------------
# Output Specification:
# Output two lines:
#   Line 1: the calculated pressure P
#   Line 2: one of the integers 1, 0, or -1
#
# ------------------------
# Sample Input 1:
# 80
#
# Sample Output 1:
# 0
# -1
#
# ------------------------
# Sample Input 2:
# 150
#
# Sample Output 2:
# 350
# 1
#
# ================================

def solve10():
    print('Enter Test Data To 11.Boiling Water')
    b = int(input())
    p = 5 * b - 400
    print(p)
    if p > 100:
        print(1)
    elif p == 100:
        print(0)
    else:
        print(-1)

#solve10()

# ================================
# USACO Bronze: Shell Game
# Source: USACO 2019 January Contest, Bronze
# Link: https://usaco.org/index.php?page=viewproblem2&cpid=891
# ================================
#
# Farmer John is playing a shell game with Bessie the cow.
# He places three shells on a table, labeled with the numbers 1, 2,
and 3.
# He then places a pebble under one of these shells.
#
# Farmer John then performs N moves.
# Each move consists of two parts:
#   1. He swaps the shells at two given positions a and b.
#   2. Bessie guesses which shell currently contains the pebble (she
guesses shell g).
#
# Your task is to determine the maximum number of correct guesses
# Bessie could have made if she had initially known which shell the
pebble was under.
#
# That is, since we don't know where the pebble started,
# you must consider all three possible initial positions (1, 2, 3)
# and determine the maximum number of times Bessie could have guessed
correctly.
#
# ----------------
# Input Format:
# ----------------
# Line 1: The integer N (1 ≤ N ≤ 100) — the number of moves.
# Lines 2..N+1: Each line contains three integers a, b, g.
#   - a and b are the two shell positions being swapped.
#   - g is Bessie's guess (the position she thinks the pebble is
under).
#
# ----------------
# Output Format:
```

```
# ----------------
# A single integer — the maximum number of correct guesses Bessie
could have made.
#
# ----------------
# Sample Input:
# ----------------
# 3
# 1 2 1
# 3 2 1
# 1 3 1
#
# ----------------
# Sample Output:
# ----------------
# 2
#
# ----------------
# Explanation:
# ----------------
# If the pebble started under shell 1 -> 1 correct guess.
# If the pebble started under shell 2 -> 2 correct guesses.
# If the pebble started under shell 3 -> 1 correct guess.
# Therefore, the maximum possible number of correct guesses is 2.
# ================================

def solve11():
   print('Enter Test Data To 12.Shell Game')
   num = int(input())
   ops = [list(map(int, input().split())) for _ in range(num)]
   an = 0
   for start in [1,2,3]:
       pearl = start
       correct = 0
       for a,b,g in ops:
           if pearl == a:
               pearl = b
           elif pearl == b:
               pearl = a
           if pearl == g:
               correct += 1
       an = max(correct,an)
```

```
    print(an)

#solve11()

# ==================================================
# 🐮 USACO 2024 January Contest, Bronze Division
# Problem: Cow College
# Link: https://usaco.org/index.php?page=viewproblem2&cpid=1377
# ==================================================

# Farmer John has just opened a new school for his cows called "Cow College"!
#
# He has surveyed N cows to determine how much each one would be willing to pay for tuition.
# The i-th cow is willing to pay at most Pi dollars.
#
# Farmer John must choose a single tuition price T (an integer).
# Every cow who is willing to pay at least T (that is, Pi ≥ T) will enroll in the college.
#
# The total revenue is then:
#     revenue = T × (number of cows whose Pi ≥ T)
#
# Farmer John wants to choose the tuition price T that maximizes his total revenue.
# If there are multiple prices that yield the same maximum revenue,
# he should choose the **smallest such T**.
#
# -----------------------------------------------
# INPUT FORMAT (from standard input):
# Line 1: The integer N (1 ≤ N ≤ 100,000)
# Next N lines: Each line contains one integer Pi (1 ≤ Pi ≤ 1,000,000,000)
#
# -----------------------------------------------
# OUTPUT FORMAT (to standard output):
# Print two integers separated by a space:
#     1️⃣ The optimal tuition price T
#     2️⃣ The maximum possible total revenue
#
# -----------------------------------------------
# SAMPLE INPUT:
# 4
# 2
# 8
# 10
# 7
#
# SAMPLE OUTPUT:
# 7 21
#
# EXPLANATION:
# If T = 2 → all 4 cows enroll → revenue = 2 × 4 = 8
# If T = 7 → cows paying [7, 8, 10] enroll → 3 × 7 = 21 ✅
# If T = 8 → cows paying [8, 10] enroll → 2 × 8 = 16
# If T = 10 → only one cow enrolls → 1 × 10 = 10
# The best choice is T = 7 with revenue = 21.
# ================================================

def solve12():
   print('Enter Test Data To 13.Cow College')
   num = int(input())
   nums = [int(input()) for _ in range(num)]
   result = 0
   an = 0
   ans = 0
   for i in range(num):
       bigger = 0
       for j in range(num):
           if nums[j] >= nums[i]:
               bigger += 1
       newresult = nums[i] * bigger
       if newresult > result:
           result = newresult
           an = nums[i]
           ans = newresult
   print(an,ans)

#solve12()

# ============================================
# CCC 2020 Senior 2: Escape Room
# Link: https://cemc.uwaterloo.ca/contests/computing/2020/stage%201/seniorEn.pdf
# ============================================

# Problem Description:
# You are given an R by C grid of positive integers.
# You start in the top-left corner (1, 1) and want to reach the bottom-right corner (R, C).
#
# You can move from a cell with integer value v to any other cell (r, c)
# such that r × c = v.
#
# For example, if the cell contains the number 6,
# then you can move to cells (1, 6), (2, 3), (3, 2), or (6, 1),
# as long as those cells exist within the boundaries of the grid.
#
# Your task is to determine whether it is possible
# to reach the bottom-right corner (R, C) starting from the top-left corner (1, 1).

# --------------------------------------------
# Input Specification:
# The first line contains two integers R and C (1 ≤ R, C ≤ 1000).
# The next R lines each contain C positive integers,
# representing the values in each cell of the grid.
#
# --------------------------------------------
# Output Specification:
# Output "yes" if it is possible to reach (R, C).
# Otherwise, output "no".
#
# --------------------------------------------
# Sample Input 1:
# 3 4
# 3 10 8 14
# 1 11 12 12
# 6 2 3 9
#
# Sample Output 1:
# yes
```

```
#
# Explanation:
# One possible sequence of moves is:
# (1,1) → (3,1) → (3,2) → (2,3) → (1,3) → (1,4) → (3,4)
# which reaches the bottom-right corner.
#
# ---------------------------------------------
# Sample Input 2:
# 2 2
# 2 4
# 6 9
#
# Sample Output 2:
# no
# =============================================

def solve13():
   R, C = map(int, input().split())
   grid = [list(map(int, input().split())) for _ in range(R)]

   stack = [(1, 1)]
   visited = set([(1, 1)])

   while stack:
       r, c = stack.pop()
       if (r, c) == (R, C):
           print("yes")
           return

       v = grid[r - 1][c - 1]

       i = 1
       while i * i <= v:
           if v % i == 0:
               r1, c1 = i, v // i
               if 1 <= r1 <= R and 1 <= c1 <= C and (r1, c1) not in
visited:
                   visited.add((r1, c1))
                   stack.append((r1, c1))

               r2, c2 = v // i, i
               if (r2, c2) != (r1, c1):
                    if 1 <= r2 <= R and 1 <= c2 <= C and (r2, c2) not in visited:
                        visited.add((r2, c2))
                        stack.append((r2, c2))
            i += 1

    print("no")

#solve13()

# ===============================================================
# CCC 2016 S2: Tandem Bicycle
# Link: https://dmoj.ca/problem/ccc16s2
# ===============================================================
#
# Problem Description:
#
# Farmers and city riders are competing in a tandem bicycle race.
# Each tandem bicycle is ridden by two riders: one from the farm and one from the city.
#
# The speed of a tandem bicycle is equal to the **maximum speed** of its two riders.
#
# You are given the speed of each rider in both groups, and an integer `type`:
#   - If `type = 1`, you must minimize the total speed of all tandem bicycles.
#   - If `type = 2`, you must maximize the total speed of all tandem bicycles.
#
# Each rider must be used exactly once.
#
# ---------------------------------------------------------------
# Input Specification:
# The first line contains an integer `type` (1 or 2).
# The second line contains an integer `n` (1 ≤ n ≤ 1000),
#   the number of riders in each group.
# The third line contains `n` space-separated integers,
#   the speeds of the first group of riders.
# The fourth line contains `n` space-separated integers,
#   the speeds of the second group of riders.
```

```
#
# ---------------------------------------------------------------
# Output Specification:
# Output one integer — the minimum or maximum possible total speed
# of all tandem bicycles, depending on the value of `type`.
#
# ---------------------------------------------------------------
# Sample Input 1:
# 1
# 3
# 5 1 4
# 6 2 4
#
# Sample Output 1:
# 12
#
# ---------------------------------------------------------------
# Sample Input 2:
# 2
# 3
# 5 1 4
# 6 2 4
#
# Sample Output 2:
# 15
#
# ---------------------------------------------------------------
# Notes:
# - For `type = 1`, both lists should be sorted in increasing order
#   to minimize the total.
# - For `type = 2`, one list should be sorted in increasing order
#   and the other in decreasing order to maximize the total.
#
# ===============================================================

def low_high(num,nums):
   for i in range(num):
       for x in range(i + 1, num):
           if nums[i] > nums[x]:
               nums[i], nums[x] = nums[x], nums[i]
   return nums
```

```python
def high_low(num,nums):
    for i in range(num):
        for x in range(i + 1,num):
            if nums[i] < nums[x]:
                nums[i], nums[x] = nums[x], nums[i]
    return nums

def solve14():
    type = int(input())
    num = int(input())
    farmers = list(map(int,input().split()))
    city = list(map(int,input().split()))

    if type == 1:
        farmers = high_low(num,farmers)
        city = high_low(num,city)
        an = 0
        for i in range(num):
            an += max(farmers[i],city[i])
        print(an)

    elif type == 2:
        farmers = high_low(num,farmers)
        city = low_high(num,city)
        an = 0
        for i in range(num):
            an += max(farmers[i],city[i])
        print(an)

#solve14()

while True:
    print('')
    print('Test: 1.Blocked Billboard | 2.Rectangle Pasture | 3.Cow Gymnastics')
    print('4.Mixing Milk | 5.Bucket Brigade | 6.Sunflowers | 7.Balanced Teams')
    print('8.Favourite Times | 9.Product Codes | 10.Time to Decompress ')
    print('11.Boiling Water | 12.Shell Game | 13.Cow College | 14.Escape Room')
    print('15.Tandem Bicycle | Exit')
```

```
   print('-'*66)
   an = input('>>>').strip()
   if an == "1" or an.lower() == "blocked billboard" or an.lower() ==
"1.blocked billboard":
       solve_blocked_billboard()
   elif an == "2" or an.lower() == "rectangle pasture" or an.lower()
== "2.rectangle pasture":
       solve1()
   elif an == "3" or an.lower() == "cow gymnastics" or an.lower() ==
"3.cow gymnastics":
       solve2()
   elif an == "4" or an.lower() == "mixing milk" or an.lower() ==
"4.mixing milk":
       solve3()
   elif an == "5" or an.lower() == "bucket brigade" or an.lower() ==
"5.bucket brigade":
       solve4()
   elif an == "6" or an.lower() == "sunflowers" or an.lower() ==
"6.sunflowers":
       solve5()
   elif an == "7" or an.lower() == "balanced teams" or an.lower() ==
"7.balanced teams":
       solve6()
   elif an == "8" or an.lower() == "favourite times" or an.lower() ==
"8.favourite times":
       solve7()
   elif an == "9" or an.lower() == "product codes" or an.lower() ==
"9.product codes":
       solve8()
   elif an == "10" or an.lower() == 'time to decompress' or
an.lower() == "10.time to decompress":
       solve9()
   elif an == "11" or an.lower() == 'boiling water' or an.lower() ==
"11.boiling water":
       solve10()
   elif an == '12' or an.lower() == 'shell game' or an.lower() ==
'12.shell game':
       solve11()
   elif an == '13' or an.lower() == 'cow college' or an.lower() ==
'13.cow college':
       solve12()
   elif an == '14' or an.lower() == 'escape room' or an.lower() ==
'14.escape room':
       solve13()
   elif an == '15' or an.lower() == 'tandem bicycle' or an.lower() ==
'15.tandem bicycle':
       solve14()
   elif an.lower() == "exit":
       break
```