한국폴리텍대학
로봇캠퍼스

재귀 호출

자료구조와 알고리즘
11주차 강의

강의 계획표 / C# 프로그래밍 2판

주	주제
1	자료구조와 알고리즘 소개
2	파이썬 기초 문법과 데이터 형식
3	선형 리스트
4	단순 연결 리스트
5	원형 연결 리스트
6	스택
7	큐
8	중간고사
9	이진 트리
10	그래프
11	**재귀 호출**
12	정렬 기본
13	정렬 고급
14	검색
15	동적 계획법
16	기말고사

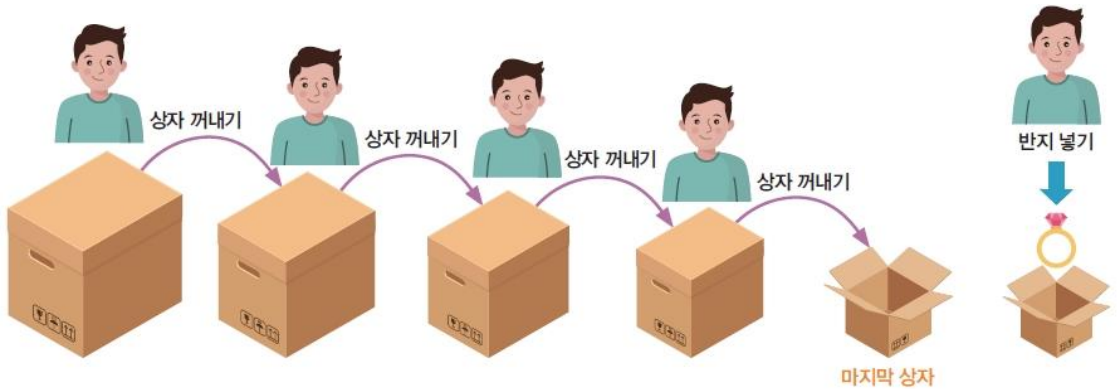
재귀 호출 기본

재귀 알고리즘이란?

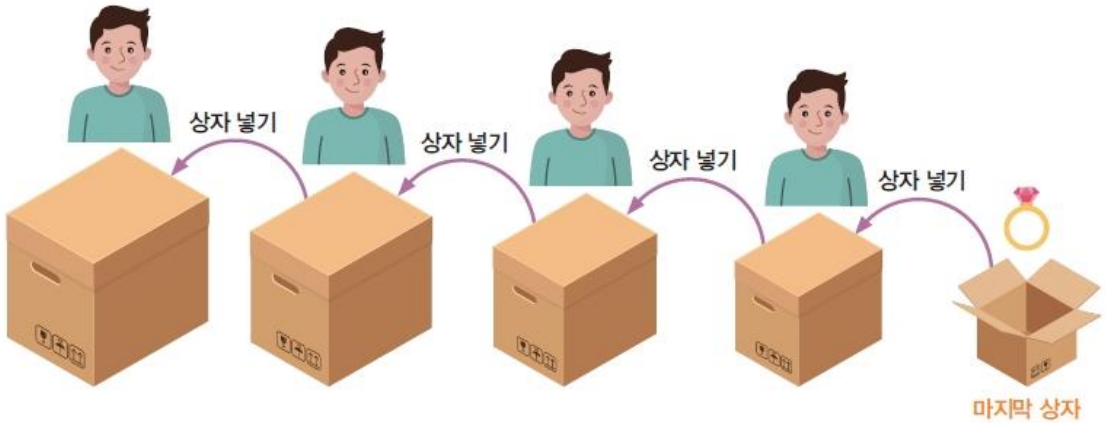
-> 양쪽에 거울이 있을 때 거울에 비친 자신이 무한 반복해서 비치는 것 또는 마트료시카 인형처럼 동일한 작동을 무한적으로 반복하는 알고리즘을 말함

재귀 호출의 개념

-> 재귀 호출(Recursion)은 자신을 다시 호출하는 것



마지막 종이 상자가 나올 때까지 여러 개 겹쳐진 상자를
꺼내 마지막 상자에 반지 넣는 예



꺼낸 순서와 반대로 종이 상자를 다시 넣는 예

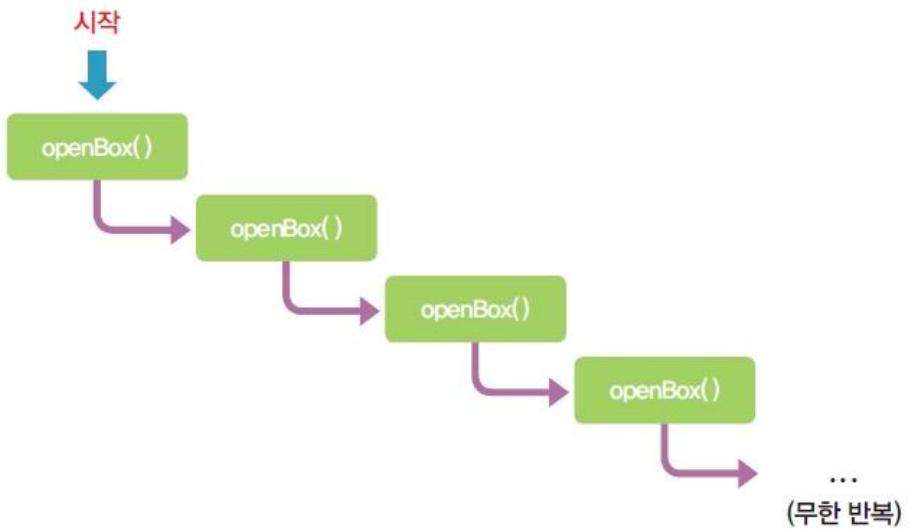
재귀 호출의 작동

-> 상자를 반복해서 여는 과정을 재귀 호출 형태로 표현

```
1  def openBox() :
2      print("종이 상자를 엽니다. ^^")
3      openBox()
4
5  openBox()     # 처음 함수를 다시 호출
```

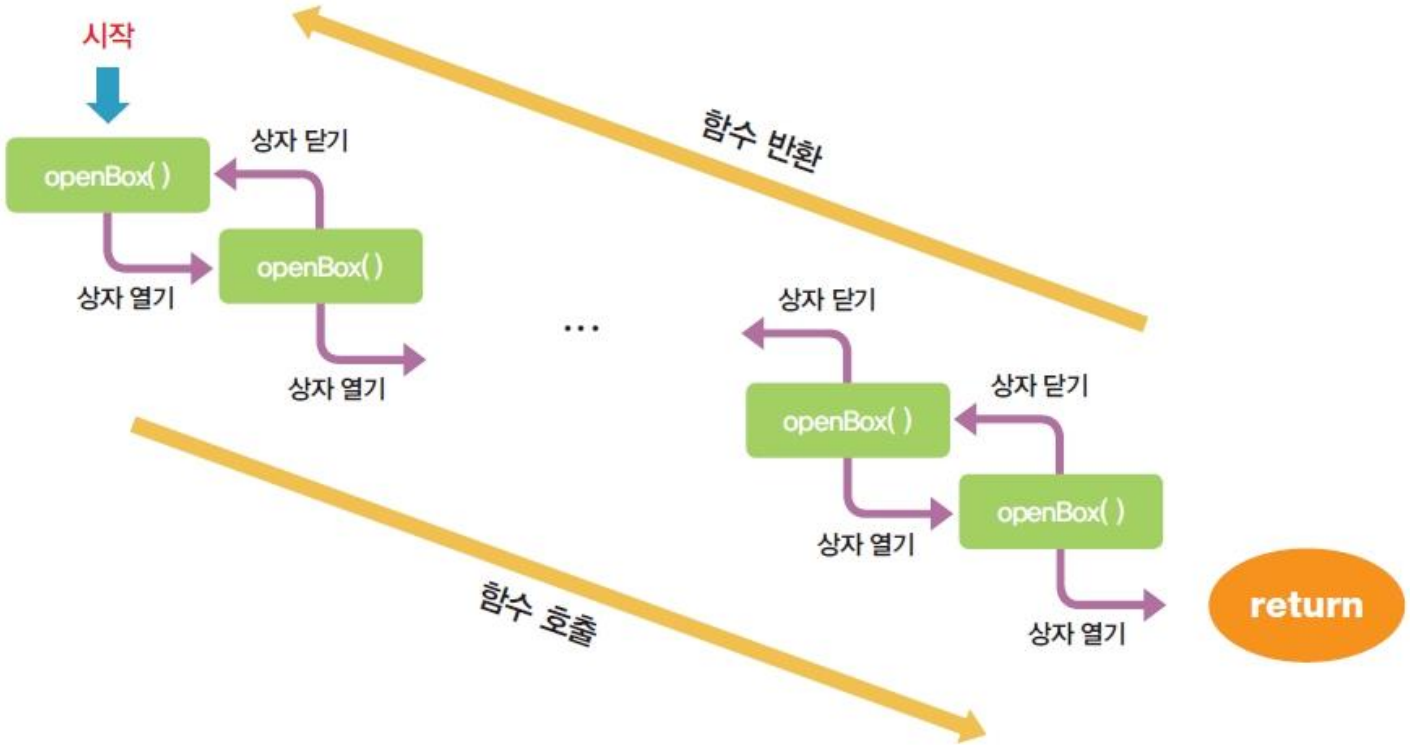
실행 결과

```
종이 상자를 엽니다. ^^
종이 상자를 엽니다. ^^
…(중략)…
Traceback (most recent call last):
  File "C:\CookData\Code10-01.py", line 5, in <module>
    openBox()    # 처음 함수를 다시 호출
  File "C:\CookData\Code10-01.py", line 3, in openBox
    openBox()
…(중략)…
  [Previous line repeated 1009 more times]
  File "C:\CookData\Code10-01.py", line 2, in openBox
    print("종이 상자를 엽니다. ^^")
RecursionError: maximum recursion depth exceeded while pickling an object
>>>
```



재귀 호출의 작동

-> 일반적인 프로그램에서는 무한 반복을 마치고 되돌아가는 조건을 함께 사용



10회 반복 후 호출한 곳으로 다시 돌아가는 조건을 사용하는 예

재귀 호출의 작동 소스 코드

```
1  def openBox() :
2      global count
3      print("종이 상자를 엽니다. ^^")
4      count -= 1
5      if count == 0 :
6          print("** 반지를 넣고 반환합니다. **")
7          return
8      openBox()
9      print("종이 상자를 닫습니다. ^^")
10
11 count = 10
12 openBox()
```

```
종이 상자를 엽니다. ^^
종이 상자를 엽니다. ^^
종이 상자를 엽니다. ^^
종이 상자를 엽니다. ^^
종이 상자를 엽니다. ^^
종이 상자를 엽니다. ^^
종이 상자를 엽니다. ^^
종이 상자를 엽니다. ^^
종이 상자를 엽니다. ^^
종이 상자를 엽니다. ^^
** 반지를 넣고 반환합니다. **
종이 상자를 닫습니다. ^^
종이 상자를 닫습니다. ^^
종이 상자를 닫습니다. ^^
종이 상자를 닫습니다. ^^
종이 상자를 닫습니다. ^^
종이 상자를 닫습니다. ^^
종이 상자를 닫습니다. ^^
종이 상자를 닫습니다. ^^
종이 상자를 닫습니다. ^^
```

실행 결과

재귀 호출의 작동 방식 이해 (호출)

```
count = 3
openBox()
```

② 함수 실행 1

```
def openBox() :
    global count
    count -= 1
    if count == 0 :
        return
    openBox()
```

① 함수 호출 1

④ 함수 실행 2

```
def openBox() :
    global count
    count -= 1
    if count == 0 :
        return
    openBox()
```

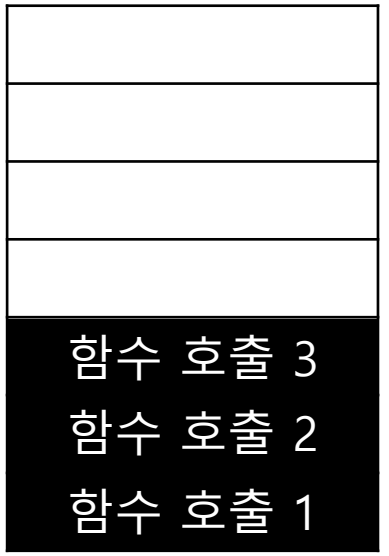
③ 함수 호출 2

⑥ 함수 실행 3

```
def openBox() :
    global count
    count -= 1
    if count == 0 :
        return
    openBox()
```

⑤ 함수 호출 3

⑦ 함수 리턴 (count가 0이 됨)



함수 호출 스택

재귀 호출의 작동 방식 이해 (리턴)

```
count = 3
openBox()
```

⑦ 프로그램 종료

```
def openBox() :
    global count
    count -= 1
    if count == 0 :
        return
    openBox()
```

⑤ 함수 리턴 (함수 종료)

```
def openBox() :
    global count
    count -= 1
    if count == 0 :
        return
    openBox()
```

③ 함수 리턴 (함수 종료)

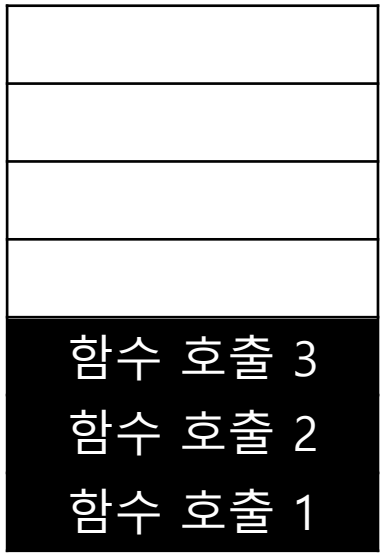
```
def openBox() :
    global count
    count -= 1
    if count == 0 :
        return
    openBox()
```

⑥ 함수 호출 1 위치로 복귀

④ 함수 호출 2 위치로 복귀

② 함수 호출 3 위치로 복귀

① 함수 리턴 (count가 0이 됨)



함수 호출 스택

재귀 호출 이해

우주선 발사 카운트다운

-> 우주선 발사를 위해 카운트하는 코드

```
import time

def countDown(n) :
    if n == 0 :
        print('발사!!')
    else :
        print(n)
        time.sleep(1)
        countDown(n-1)

countDown(5)
```

실행 결과

```
5
4
3
2
1
발사!!
```

재귀 호출을 사용한 1부터 10까지 숫자 합계 내기
-> 반복문을 이용한 구현

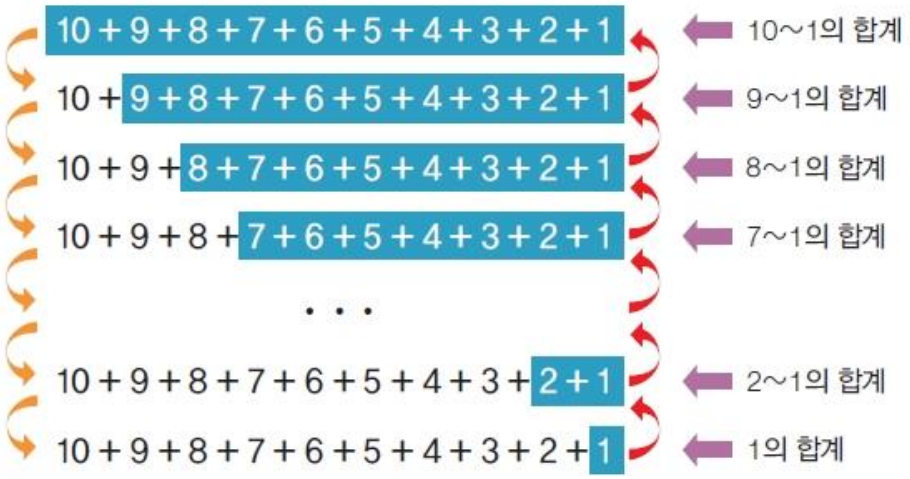
```
sumValue = 0
for n in range(10, 0, -1) :
   sumValue += n
print("10+9+…+1 = ", sumValue)
```

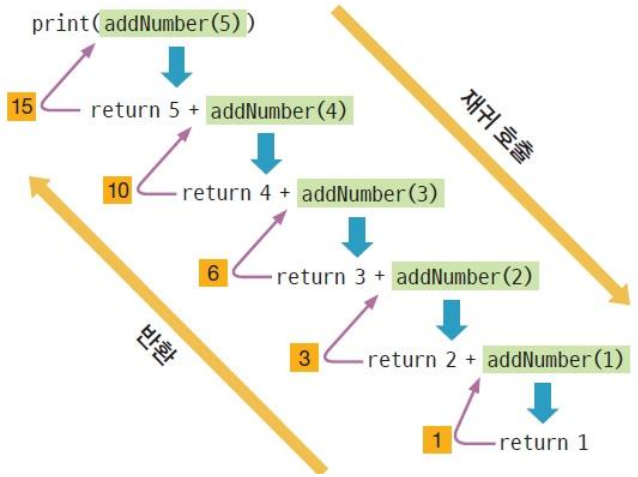
실행 결과

```
10+9+…+1 = 55
```

재귀 호출을 사용한 1부터 10까지 숫자 합계 내기
-> 재귀 호출을 이용한 구현

```
def addNumber(num) :
    if num <= 1 :
        return 1
    return num + addNumber(num-1)

print(addNumber(10))
```





(1단계) 6행의 addNumber(10) 호출
(2단계) 4행의 10 + addNumber(9) 호출
(3단계) 4행의 9 + addNumber(8) 호출
(4단계) 4행의 8 + addNumber(7) 호출
(5단계) 4행의 7 + addNumber(6) 호출
(6단계) 4행의 6 + addNumber(5) 호출
(7단계) 4행의 5 + addNumber(4) 호출
(8단계) 4행의 4 + addNumber(3) 호출
(9단계) 4행의 3 + addNumber(2) 호출
(10단계) 4행의 2 + addNumber(1) 호출
(11단계) 3행의 1 반환
(10단계) 4행의 2 + 1(=3) 반환
(9단계) 4행의 3 + 3(=6) 반환
(8단계) 4행의 4 + 6(=10) 반환
(7단계) 4행의 5 + 10(=15) 반환
(6단계) 4행의 6 + 15(=21) 반환
(5단계) 4행의 7 + 21(=28) 반환
(4단계) 4행의 8 + 28(=36) 반환
(3단계) 4행의 9 + 36(=45) 반환
(2단계) 4행의 10 + 45(=55) 반환
(1단계) 6행의 55 출력

연습 문제

Code 를 수정해서 두 수를 입력받고 두 숫자 사이의 합계를 구하는 코드를 작성하자. 단 입력하는 숫자는 작은 숫자 또는 큰 숫자 중 어느 것을 먼저 입력해도 같다.

실행 결과

```
숫자1-->1
숫자2-->10
55

숫자1-->10
숫자2-->1
55
```

재귀 호출을 사용한 팩토리얼 계산하기

-> 반복문을 이용한 구현

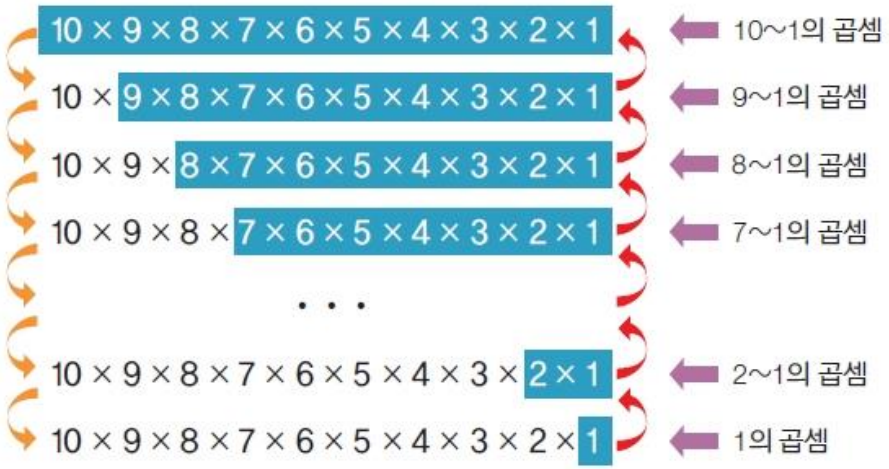
```
factValue = 1   # 곱셈이므로 초깃값을 1로 설정
for n in range(10, 0, -1) :
    factValue *= n
print("10*9*…*1 = ", factValue)
```

실행 결과

```
10*9*…*1 = 3628800
```

-> 재귀 함수를 이용한 구현

```
def factorial(num) :
    if num <= 1 :
        return 1
    return num * factorial(num-1)

print('\n10! = ', factorial(10))
```



재귀 호출을 사용한 팩토리얼 계산하기 소스코드와 단계별 작동

```python
def factorial(num) :
    if num <= 1 :
        print('1 반환')
        return 1
    print("%d * %d! 호출" % (num, num-1))
    retVal = factorial(num-1)

    print("%d * %d!(=%d) 반환" % (num, num-1, retVal))
    return num * retVal

print('\n5! = ', factorial(5))
```

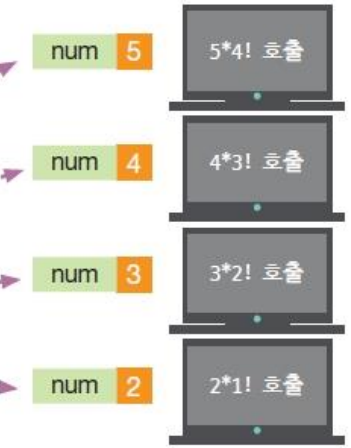
재귀 호출을 사용한 팩토리얼 계산하기 소스코드와 단계별 작동

1 A부분에서 11행의 factorial(5)를 호출한다.

```
def factorial(num) :                                    A
    if num <= 1 :
        print('1 반환')
        return 1
    print("%d * %d! 호출" % (num, num-1))

B   retVal = factorial(num-1)

    print("%d * %d!(=%d) 반환" % (num, num-1, retVal))
    return num * retVal

print('\n5! = ', factorial(5))
```

2 ~ 5 num이 5, 4, 3, 2인 상태에서 A 부분의 5~6행 실행(재귀 함수 1~4회 호출)

```
def factorial(num) :                                    A
    if num <= 1 :
        print('1 반환')
        return 1
    print("%d * %d! 호출" % (num, num-1))
        factorial(num-1)
```



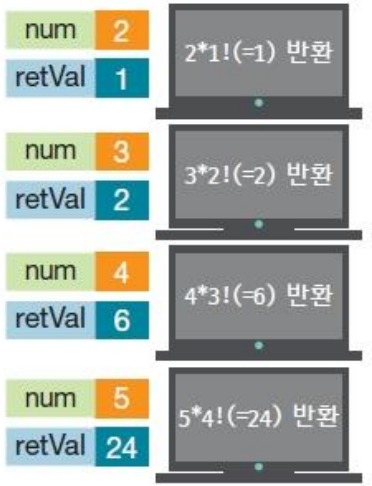
재귀 호출을 사용한 팩토리얼 계산하기 소스코드와 단계별 작동

6 A부분에서 11행의 factorial(5)를 호출한다.

```
def factorial(num) :                                    A
    if num <= 1 :
        print('1 반환')
        return 1
    print("%d * %d! 호출" % (num, num-1))
    factorial(num-1)
```

반환된 5 ~ 2 num이 2, 3, 4, 5인 상태에서 B 부분의 8~9행 실행

```
retVal                                                  B
    print("%d * %d!(=%d) 반환" % (num, num-1, retVal))
    return num * retVal
```



재귀 호출을 사용한 팩토리얼 계산하기 소스코드와 단계별 작동

반환된 1 처음 호출한 factorial(5)가 반환되어 최종 5!인 120 출력

```
def factorial(num) :                                    A
    if num <= 1 :
        print('1 반환')
        return 1
    print("%d * %d! 호출" % (num, num-1))

B   retVal = factorial(num-1)

    print("%d * %d!(=%d) 반환" % (num, num-1, retVal))
    return num * retVal

print('\n5! = ', factorial(5))
```

별 모양 출력하기

-> 입력한 숫자만큼 차례대로 별 모양을 출력하는 코드

```
def printStar(n) :
    
    
printStar(5)
```

★
★★
★★★
★★★★
★★★★★

구구단 출력하기

-> 2단부터 9단까지 구구단을 출력하는 코드

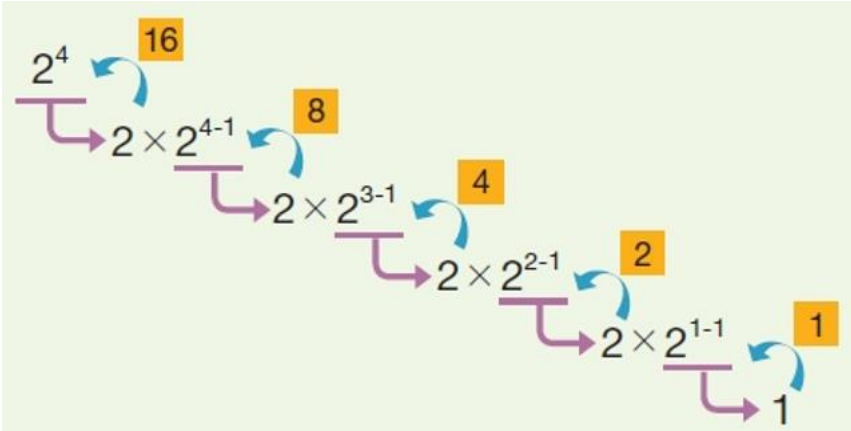
```
def gugu(dan, num) :
    print("%d x %d = %d" % (dan, num, dan*num))
    if num < 9:
        gugu(dan, num+1)

for dan in range(2,10) :
    print("## %d단 ##" % dan)
    gugu(dan, 1)
```

구구단이 우측과 같이 출력되도록 코드를 고쳐보자

```
2x1= 2  3x1= 3  4x1= 4  5x1= 5  6x1= 6  7x1= 7  8x1= 8  9x1= 9
2x2= 4  3x2= 6  4x2= 8  5x2=10  6x2=12  7x2=14  8x2=16  9x2=18
2x3= 6  3x3= 9  4x3=12  5x3=15  6x3=18  7x3=21  8x3=24  9x3=27
2x4= 8  3x4=12  4x4=16  5x4=20  6x4=24  7x4=28  8x4=32  9x4=36
2x5=10  3x5=15  4x5=20  5x5=25  6x5=30  7x5=35  8x5=40  9x5=45
2x6=12  3x6=18  4x6=24  5x6=30  6x6=36  7x6=42  8x6=48  9x6=54
2x7=14  3x7=21  4x7=28  5x7=35  6x7=42  7x7=49  8x7=56  9x7=63
2x8=16  3x8=24  4x8=32  5x8=40  6x8=48  7x8=56  8x8=64  9x8=72
2x9=18  3x9=27  4x9=36  5x9=45  6x9=54  7x9=63  8x9=72  9x9=81
```

N제곱 계산하기

```
tab = ''
def pow(x, n) :
    global tab
    tab += '    '
    if  n == 0 :
        return 1
    print(tab+"%d*%d^(%d-%d)" % (x, x, n, 1))
    return x * pow (x, n-1)

print('2^4')
print('답 -->', pow(2, 4))
```



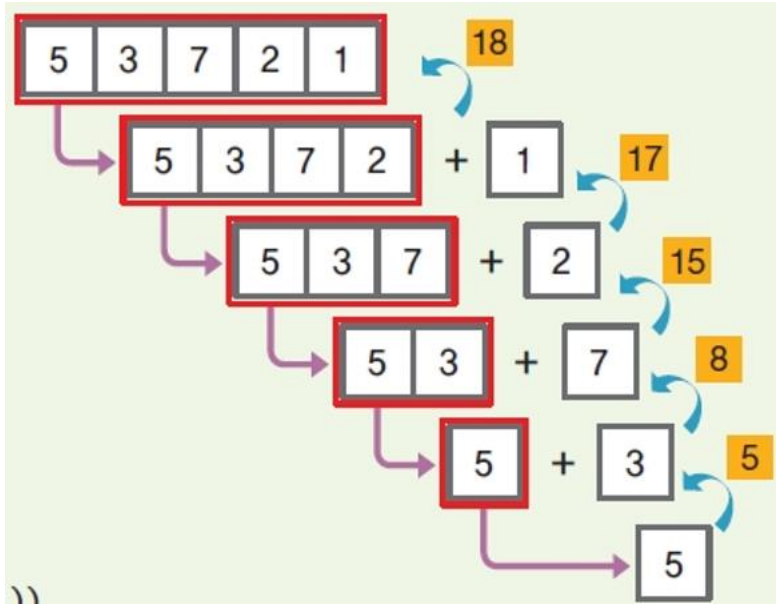
실행 결과

```
2^4
        2*2^(4-1)
                2*2^(3-1)
                        2*2^(2-1)
                                2*2^(1-1)
답 --> 16
```

배열의 합 계산하기

-> 랜덤하게 생성한 배열의 합계를 구하는 코드

```
import random

def arySum(arr, n) :
    if n <= 0 :
        return arr[0]
    return arySum(arr, n-1) + arr[n]

ary = [random.randint(0, 255) for _ in
range(random.randint(10, 20))]
print(ary)
print('배열 합계 -->', arySum(ary, len(ary)-1))
```

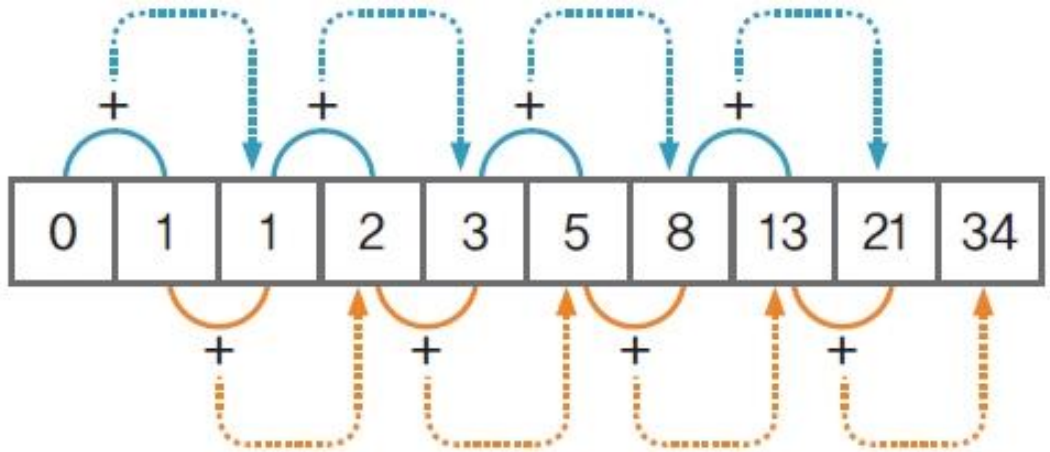


실행 결과

```
[91, 183, 8, 208, 38, 244, 54, 79, 118, 63, 52, 148, 216, 189, 85, 250, 34, 242, 137, 150]
배열 합계 --> 2589
```

피보나치 수

```
def fibo(n) :
    if n == 0 :
        return 0
    elif n == 1 :
        return 1
    else :
        return fibo(n-1) + fibo(n-2)

print('피보나치 수 --> 0 1 ', end='')
for i in range(2, 20) :
    print(fibo(i), end=' ')
```



실행 결과

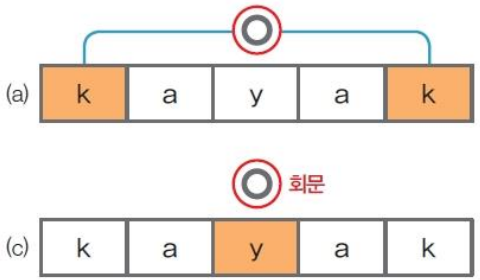
```
피보나치 수 --> 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```

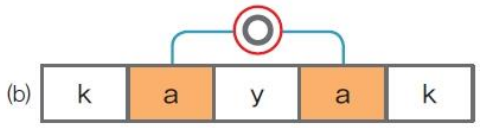
재귀 호출 응용

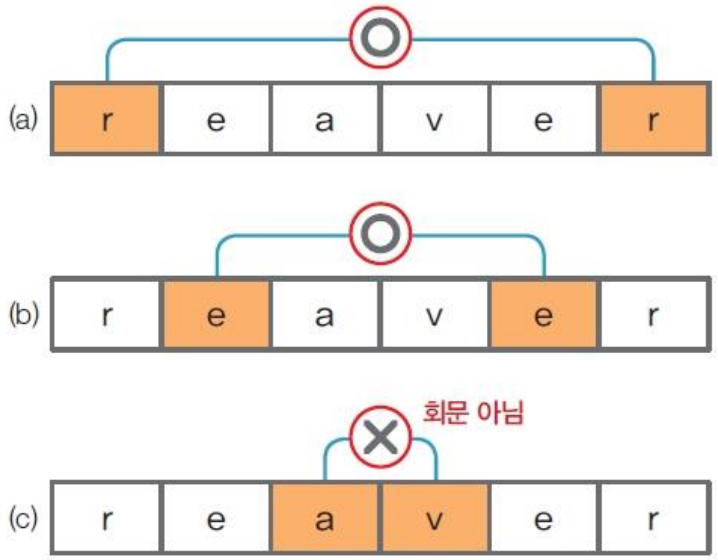
회문 판단하기

-> 회문(Palindrome)은 앞에서부터 읽든 뒤에서부터 읽든 동일한 단어나 문장을 의미
-> 대, 소문자는 구분하지 않고 공백이나 특수문자는 제외
-> 첫 글자와 마지막 글자, 그 다음 글자들을 비교해 나가면서 다른 글자가 발견되는 경우 회문이 아니라고 판단
-> 비교하지 않은 글자가 한 글자가 남는 경우는 무시

```
level
kayak
radar
Borrow or rob
I prefer pi
기러기
일요일
주유소의 소유주
다 큰 도라지일지라도 큰다
야 너 이번 주 주번이 너야
야 이 달은 밝은 달이야
마지막 날 날 막지 마
```







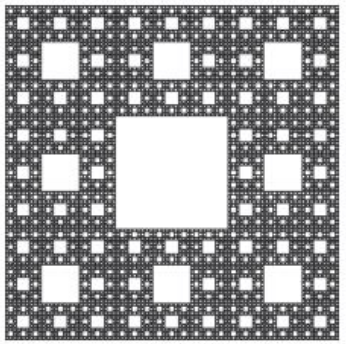
회문 판단하기 코드

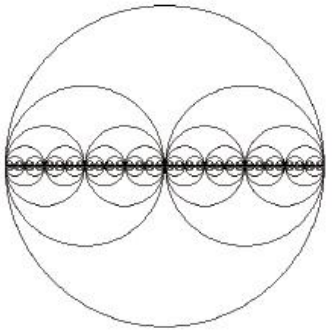
```python
## 함수 선언 부분 ##
def palindrome(pStr) :
    if len(pStr) <= 1 :
        return True

    if pStr[0] != pStr[-1] :
        return False

    return palindrome(pStr[1:len(pStr)-1])

## 전역 변수 선언 부분 ##
strAry = ["reaver", "kayak", "Borrow or rob", "주유소의 소유주", "야 너 이번주 주번이 너야", "살금 살금"]

## 메인 코드 부분 ##
for testStr in strAry :
    print(testStr, end = '--> ' )
    testStr = testStr.lower().replace(' ','')
    if palindrome(testStr) :
        print('O')
    else :
        print('X')
```

프랙탈 그리기

-> 프랙탈(Fractal)은 작은 조각이 전체와 비슷한 기하학적인 형태를 의미(자기 유사성)

-> 자기 유사성(Self Similarity)은 부분을 확대하면 전체와 동일한 또는 닮은 꼴의 모습을 나타내는 성질이 있음

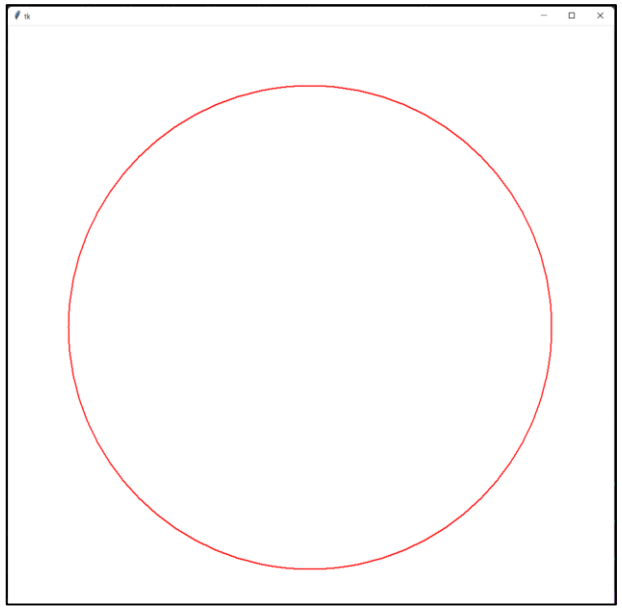




프랙탈 그리기 코드

-> 1000×1000 크기의 윈도 창을 만들고 중앙에 반지름 400 크기의 원을 그리는 코드
-> tkinter 라이브러리 : 윈도 창과 점, 선, 원 등을 그릴 수 있음

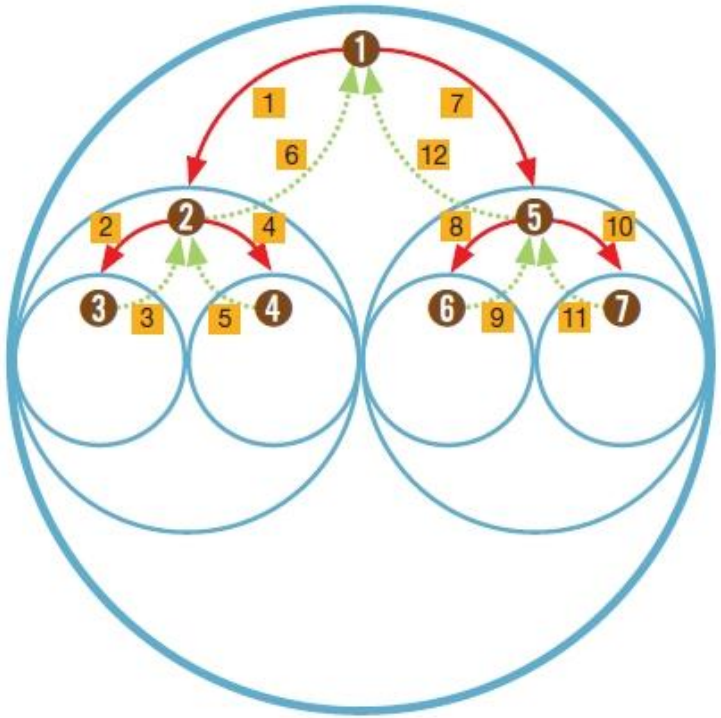
```
from tkinter import *

window = Tk()
canvas = Canvas(window, height=1000, width=1000, bg='white')
canvas.pack()

cx=1000//2
cy=1000//2
r = 400
canvas.create_oval(cx-r, cy-r, cx+r, cy+r, width=2, outline="red")

window.mainloop()
```



재귀 호출을 이용한 원 도형 프랙탈 그리기 코드

-> radius 변수에 재귀 함수를 탈출하는 조건을 주고 원을 그려 나감

```
from tkinter import *

## 함수 선언 부분 ##
def drawCircle(x, y, r) :
    global count
    count += 1
    canvas.create_oval(x-r, y-r, x+r, y+r)
    canvas.create_text(x, y-r, text=str(count), font=('', 30))
    if r >= radius/2 :
        drawCircle(x-r//2, y, r//2)
        drawCircle(x+r//2, y, r//2)

## 전역 변수 선언 부분 ##
count = 0
wSize = 1000
radius = 400

## 메인 코드 부분 ##
window = Tk()
canvas = Canvas(window, height=wSize, width=wSize, bg='white')

drawCircle(wSize//2, wSize//2, radius)

canvas.pack()
window.mainloop()
```

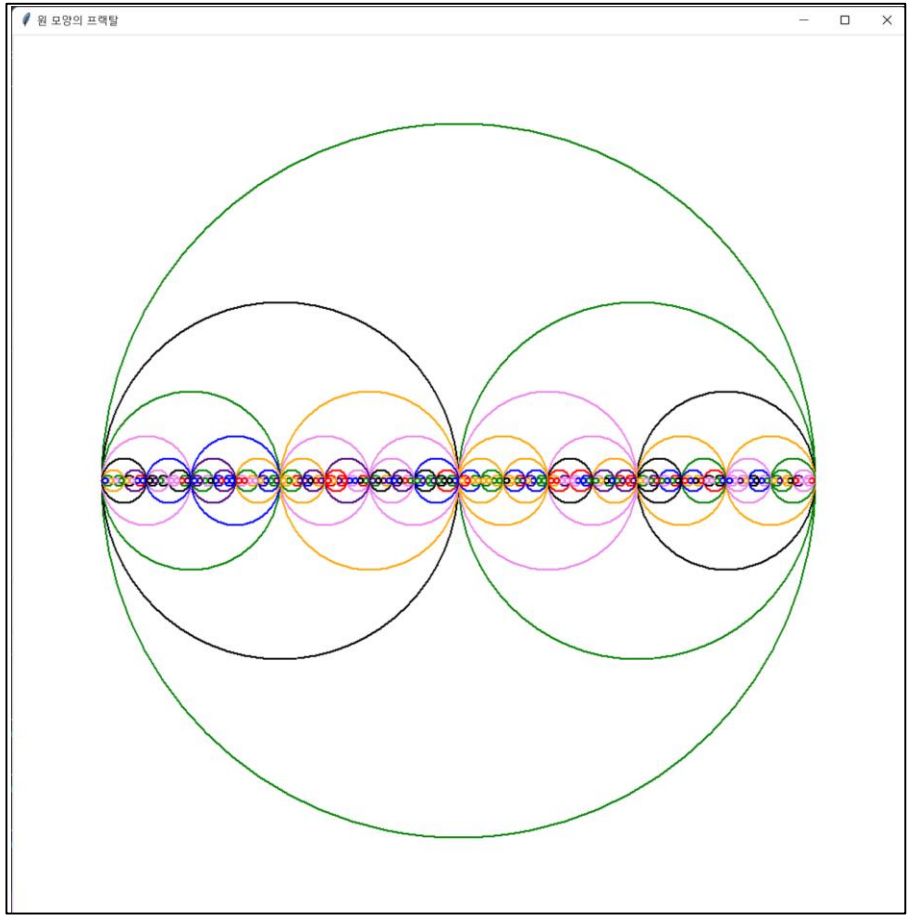


원을 더 많이 그리도록 코드를 변경해보자

원 도형 전체 프랙탈 그리기 응용 코드

-> 반지름이 5 이상일 때 까지 반복하고 색상을 다양하게 변경

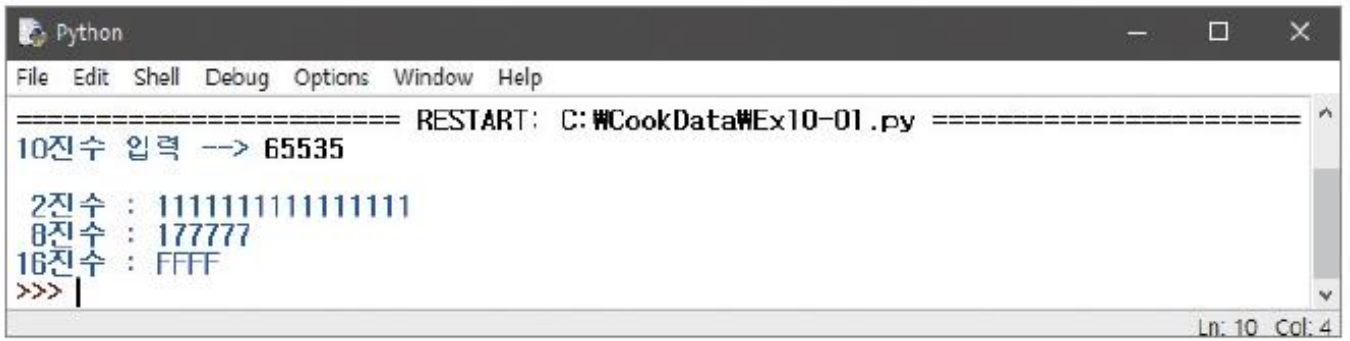
```
from tkinter import *
import random

## 함수 선언 부분 ##
def drawCircle(x, y, r) :
    canvas.create_oval(x-r, y-r, x+r, y+r, width=2,
outline=random.choice(colors))
    if r >= 5 :
        drawCircle(x+r//2, y, r//2)
        drawCircle(x-r//2, y, r//2)

## 전역 변수 선언 부분 ##
colors = ["red", "green", "blue", "black", "orange", "indigo", "violet"]
wSize = 1000
radius = 400

## 메인 코드 부분 ##
window = Tk()
window.title("원 모양의 프랙탈")
canvas = Canvas(window, height=wSize, width=wSize, bg='white')

drawCircle(wSize//2, wSize//2, radius)

canvas.pack()
window.mainloop()
```



진수 변환하기

예제 설명 10진수 정수를 입력하면, 2진수/8진수/16진수로 변환되어 출력되는 프로그램을 재귀 함수를 이용하여 작성한다.

실행 결과



진수 변환하기 코드

```
## 함수 선언 부분 ##
def notation(base, n):
    if n < base :
        print(numberChar[n], end = ' ')
    else :
        notation(base, n // base)
        print(numberChar[n % base], end = ' ')

## 전역 변수 선언 부분 ##
numberChar = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
numberChar += ['A', 'B', 'C', 'D', 'E', 'F']

## 메인 코드 부분 ##
number = int(input('10진수 입력 -->'))
print('\n 2진수 : ', end = ' ')
notation(2, number)
print('\n 8진수 : ', end = ' ')
notation(8, number)
print('\n16진수 : ', end = ' ')
notation(16, number)
```

10진수를 2, 8, 16진수로의 변환은
10진수의 수를 변환하고자 하는 진수로
나누고 나머지를 기록해 나간 후
기록된 나머지를 역순으로 배치함

실행 결과

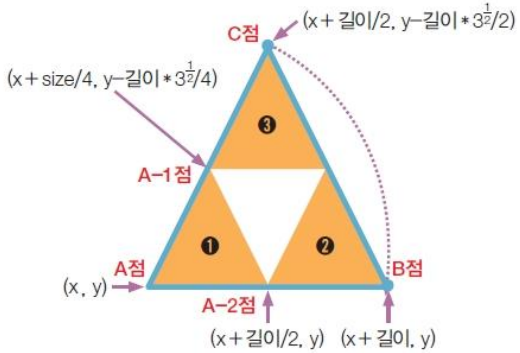
```
10진수 입력 -->10

 2진수 :  1 0 1 0
 8진수 :  1 2
16진수 :  A
```

시에르핀스키 삼각형 그리기

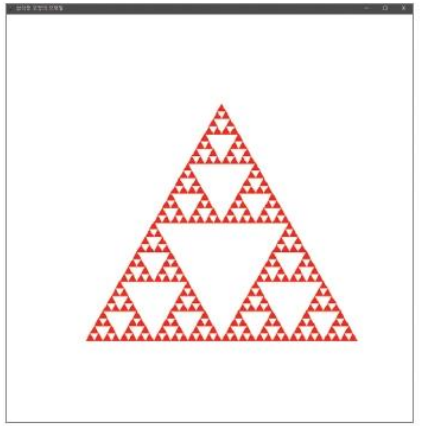
예제 설명

시에르핀스키(Sierpinski) 삼각형은 다음과 같이 큰 삼각형 안에 다시 정삼각형 3개를 균등하게 생성한다. 변의 길이와 왼쪽 아래 시작점(x, y)이 정해지면 큰 삼각형은 A점, B점, C점을 연결하면 된다. 그리고 다시 안쪽 삼각형 3개는 A점, A−1점, A−2점이 시작점이 되고 변의 길이는 1/2이 된다. 이것을 재귀 호출로 반복하면 프랙탈 모양이 된다. 각 위치를 구하는 수식은 그림에 표현했다.

피타고라스 정리



실행 결과



시에르핀스키 삼각형 그리기 코드

-> create_polygon : 다각형을 그리는 함수, 아래 코드는 삼각형을 그리기 위해 세 꼭지점을 매개변수로 전달

```
from tkinter import *

## 함수 선언 부분 ##
def drawTriangle(x, y, size) :
    if size >= 30 :
        drawTriangle(x, y, size/2)              # 왼쪽 아래 작은 삼각형
        drawTriangle(x+size/2, y, size / 2)         # 오른쪽 아래 작은 삼각형
        [redacted]                                  # 위쪽 작은 삼각형
    else :
        canvas.create_polygon (x, y, x + size, y, x + size / 2, y - size*(3 ** 0.5) / 2, fill = 'red', outline = "red")

## 전역 변수 선언 부분 ##
wSize = 1000
radius = 400

## 메인 코드 부분 ##
window = Tk()
window.title("삼각형 모양의 프랙탈")
canvas = Canvas(window, height = wSize, width = wSize, bg = 'white')

drawTriangle(wSize/5, wSize/5*4, wSize*2/3)

canvas.pack()
window.mainloop()
```

다음 강의 예고

정렬 기본

- 정렬 기본
- 기본 정렬 알고리즘의 원리와 구현
- 기본 정렬 알고리즘의 응용

Q&A

감사합니다.