Teknik Rekursif

Teknik pemrograman Rekursif

Beberapa permasalahan tertentu terkadang membutuhkan penyelesaian dengan cara perulangan, misalkan permasalahan pengurutan, perhitungan deret, dan sebagainya.
Terdapat dua metode untuk menyelesaikan permasalahan tersebut, yaitu :

1. Iteratif
2. Rekursif
3. Implementasi Rekursif
4. Visualisasi Fraktal

Iteratif

Iteratif merupakan penyelesaian permasalahan dengan perulangan, menggunakan syntax `for` atau `while` (pada Python).

Misalkan permasalahan perhitungan faktorial suatu bilangan, yaitu : $5! = 5 \times 4 \times 3 \times 2 \times 1$, atau
$n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 1$,
maka perhitungan faktorial suatu bilangan dapat dilakukan dengan cara iterasi, yaitu mengmenghitung operasi perkalian mulai dari bilangan tersebut sampai dengan satu.

Code

Berikut adalah code untuk perhitungan faktorial suatu bilangan dengan menggunakan cara iteratif.
Pada code ini, terdapat iterasi while, yang menghitung perkalian, mulai dari n, sampai dengan 2.

In [1]:
```
def factorialIteratif(bilangan):
    if bilangan <=1:
        return(1)
    else:
        temp=bilangan
        hasil=1
        while temp>1:
            hasil=hasil*temp
            temp=temp-1
        return(hasil)
```

In [2]:
```
print(factorialIteratif(4))
```

```
24
```

Kembali ke Menu Awal

Suatu permasalahan akan lebih mudah dipecahkan jika permasalahan tersebut berukuran kecil. Misalkan pengurutan data akan lebih mudah dilakukan jika data yang diurutkan hanya terdiri dari dua buah data saja, sehingga pengurutan dapat dilakukan dengan membandingkan dua data tersebut saja.
Rekursif merupakan suatu teknik pemrograman yang memecah permasalahan menjadi permasalahan dengan ukuran yang lebih kecil dan lebih kecil lagi, sehingga permasalahan dengan ukuran kecil tersebut dapat dengan mudah dipecahkan. Teknik ini dapat dilakukan dengan cara memanggil fungsi itu sendiri.

Jika pada contoh sebelumnya, untuk menghitung suatu bilangan dapat dilakukan dengan menggunakan cara iterasi, maka berikut ini adalah penyelesaian perhitungan faktorial dengan cara rekursif.

Misalkan hitung 5!, maka dapat dihitung dengan cara sebagai berikut,

$$\begin{aligned} 5! &= 5 \times 4 \times 3 \times 2 \times 1 \\ &= 5 \times 4! \end{aligned}$$

dimana

$$\begin{aligned} 4! &= 4 \times 3 \times 2 \times 1 \\ &= 4 \times 3! \end{aligned}$$

dimana

$$\begin{aligned} 3! &= 3 \times 2 \times 1 \\ &= 3 \times 2! \end{aligned}$$

dimana

$$\begin{aligned} 2! &= 2 \times 1 \\ &= 1 \times 1! \end{aligned}$$

dimana $1! = 1$

atau jika dibuat pola secara umum, maka $n! = n \times (n-1)!$

Code

Berikut adalah code untuk perhitungan faktorial dengan cara rekursif

In [3]:
```
def factorialRekursif(bilangan):
    if bilangan <=1:
        return(1)
    else:
        return(bilangan * factorialRekursif(bilangan-1))
```

In [5]:
```
data=factorialRekursif(4)
print(data)
```

```
24
```

Dari fungsi tersebut dapat dilihat bahwa teknik pemrograman rekursif ini melibatkan pemanggilan fungsinya itu sendiri dengan argument atau parameter yang berukuran lebih kecil. Berikut beberapa aturan dalam teknik pemrograman rekursif:

- Fungsi rekursif harus memiliki *base case*, *base case* inilah yang berfungsi untuk menghentikan pemanggilan terus menerus fungsi rekursif
- fungsi rekursif harus memiliki sintaks yang dapat merubah state dari permasalahan, sehingga semakin lama solusi permasalahan menuju *base case* yang sudah dibuat
- Fungsi rekursif harus memanggil dirinya sendiri

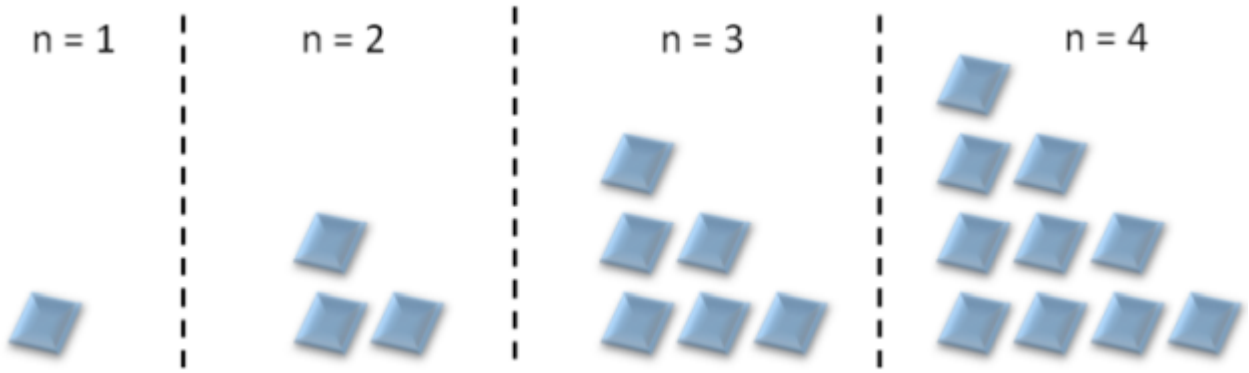
Kembali ke Menu Awal

Implementasi Rekursif

Banyak permasalahan yang dapat diselesaikan dengan teknik pemrograman rekursif ini. Berikut contoh-contoh implementasi rekursif.

Triangular Numbers

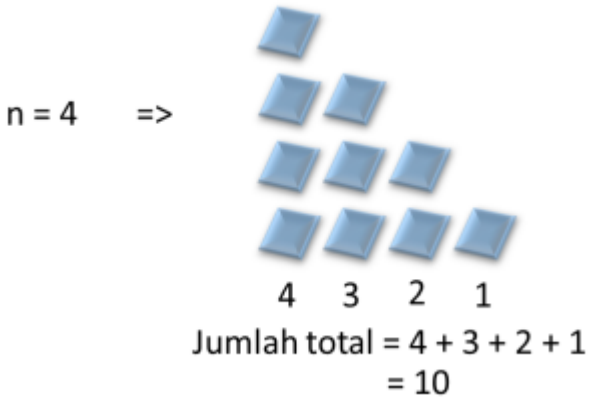
Terdapat deret bilangan berikut : 1, 3, 6, 10, 15, 21, 28, 36, ... Bagaimanakah pola bilangan tersebut ?
Deret bilangan tersebut dikenal dengan nama **Triangular Numbers**, yang dapat diilustrasikan seperti Gambar 1 berikut.



Gambar 1. Triangular Numbers

Bagaimana menentukan bilangan ke-n dari deret tersebut ?

Ilustrasi perhitungan deret keempat dari **triangular numbers** dapat dilihat pada Gambar 2 berikut:



Gambar 2. Perhitungan Triangular Numbers

Dapat dilihat pada Gambar tersebut bahwa kolom pertama memiliki 4 kotak, kolom kedua memiliki 3 kotak, kolom ketiga memiliki 2 kotak, dan kolom terakhir memiliki 1 kotak, jumlah kotak total adalah 10, oleh karena itu deret ke-empat dari triangular numbers adalah 10.
Berikut fungsi iteratif untuk mencari deret ke-n dari triangular numbers

Code

Berikut adalah code untuk menghitung *triangular numbers* melalui teknik iteratif

In [6]:
```
def triangularNumbers1(n):
    total=0
    temp=n
    while temp>=1:
        total+=temp
        temp-=1
    return(total)
```

In [7]:
```
print(triangularNumbers1(2))
```

```
3
```

Code

Untuk mencari deret ke-n dari triangular numbers, dapat juga digunakan fungsi rekursif seperti permasalahan perhitungan faktorial sebelumnya. *Base case* dari triangular numbers ini adalah ketika deret ke-1 yang bernilai 1. Berikut fungsi rekursif dari triangular numbers.

In [0]:
```
def triangularNumbers2(n):
    if n==1:
        return(1)
    else:
        return (n + triangularNumbrs2(n-1))
```

In [0]:
```
print(triangularNumbers1(8))
```

Konversi Bilangan

Bilangan integer dikenal juga sebagai bilangan dengan *base* 10 (desimal), sehingga memiliki kemungkinan character angka dari 0 s.d 9. Sedangkan *base* 2 atau yang dikenal nama bilangan biner, hanya memiliki kemungkinan character angka 0 dan 1 saja. Base 8 atau octal, hanya memiliki character 0 s.d 7, sedangkan base 16 atau hexa, memiliki character angka mulai dari 0 s.d 9 dan 'A','B','C','D','E', dan 'F'. Masing-masing bilangan ini dapat dikonversikan menjadi bilangan base lain, akan tetapi yang sering digunakan adalah konversi bilangan desimal ke bilangan base yang lain, atau dari base 2, 8, 16 ke bilangan desimal. Berikut contoh konversi bilangan desimal ke bilangan base 2, 8, atau 16.

Bilangan desimal 8 menjadi bilangan biner '1000'. Untuk merubah bilangan desimal 8 menjadi biner, diperlukan operasi pembagian dan modulo dari bilangan desimal tersebut.

Code

Berikut fungsi rekursif untuk konversi bilangan desimal menjadi bilangan base yang lain.

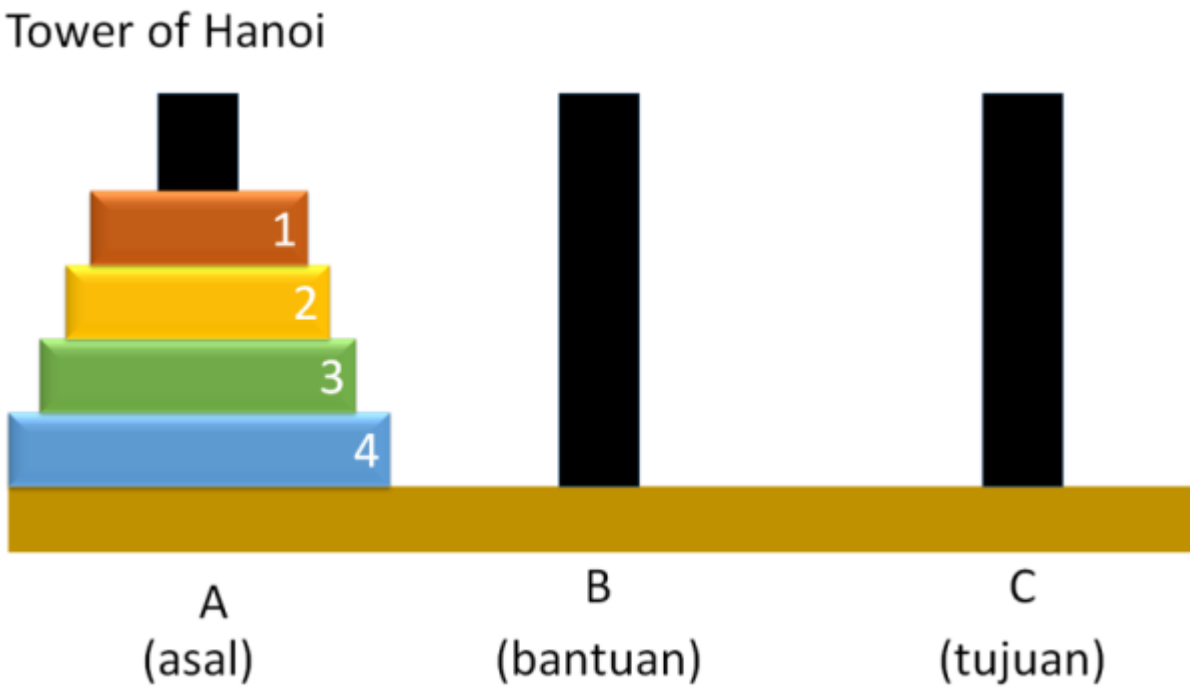
In [0]:
```
def konversiDesimal(bilangan,base):
    charBilangan='0123456789ABCDEF'
    if bilangan<base:
        return(charBilangan[bilangan])
    else:
        temp=bilangan//base
        ind=bilangan % base
        print(type(temp),temp,type(ind),ind)
        return(konversiDesimal(temp,base)+charBilangan[ind])
```

In [0]:
```
konversiDesimal(5,2)
```

Towers of Hanoi

Towers of Hanoi ini diinspirasi oleh cerita mitos dari sebuah kuil di India. Pada saat itu, pendeta memerintahkan para pekerjanya atau pendeta muda untuk memindahkan 64 piringan emas dengan ukuran yang berbeda-beda dari satu tiang ke tiang lainnya. Syarat memindahkan piringan emas ini adalah :

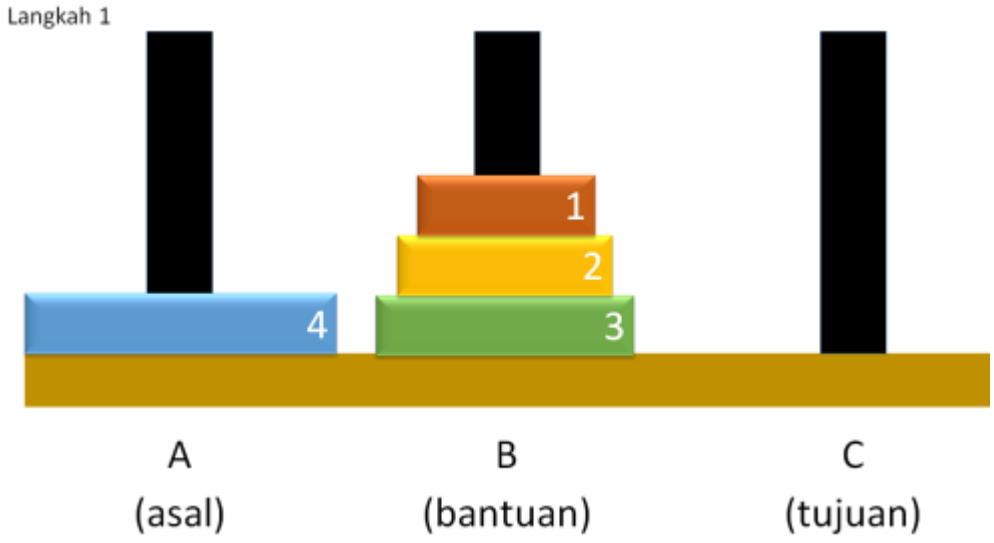
- piringan emas harus dipindahkan satu persatu
- piringan emas berukuran kecil harus berada diatas piringan emas berukuran besar. Piringan emas dan tiang, dapat diilustrasikan pada Gambar 3 berikut :



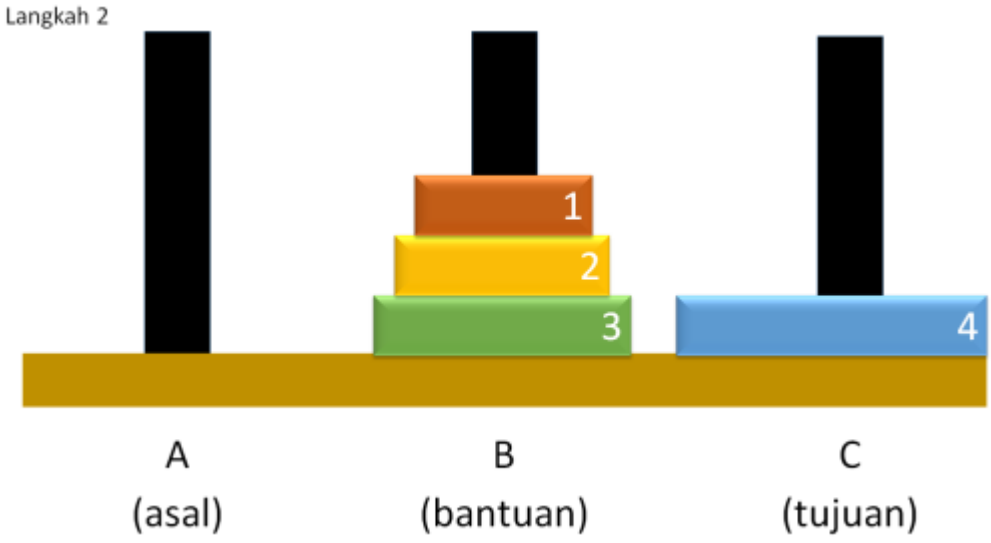
Gambar 3. Tower of Hanoi

Mitos mengatakan bahwa, ketika tugas ini selesai, maka kuil akan lenyap dan dunia akan hilang. Tetapi ini hanya mitos saja, dan untuk memindahkan piringan emas satu persatu itu membutuhkan waktu milyaran tahun. Pemindahan piringan emas tersebut akan mudah, jika hanya 1 buah piringan saja yang akan dipindahkan.

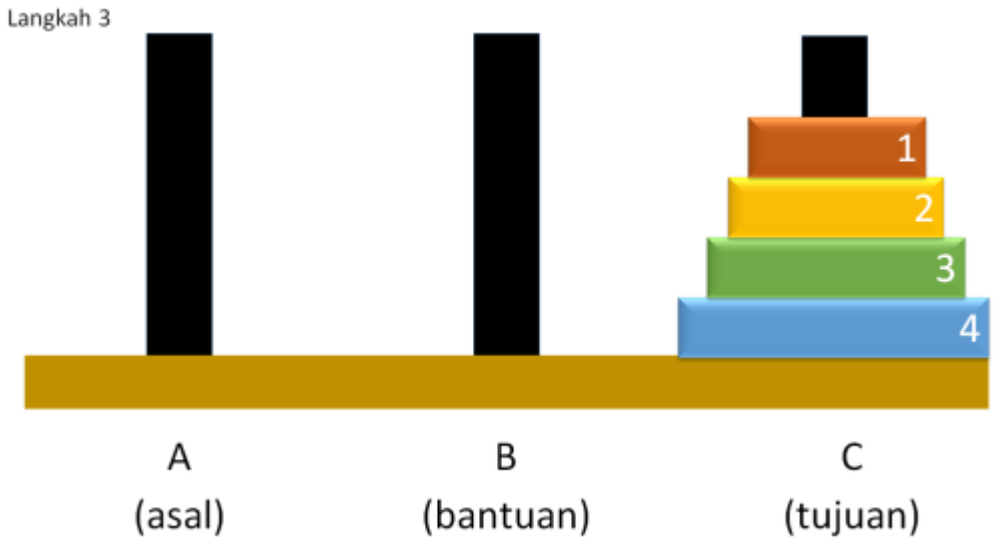
Misalkan, terdapat 4 buah piringan terdapat di tiang 'A' dan akan dipindahkan ke tiang 'C', dan ada tiang 'B' sebagai bantuan. Untuk memindahkan 4 piringan tersebut dari tiang A ke tiang C, bukanlah hal yang mudah. Akan tetapi jika diasumsikan, bahwa 3 buah piringan teratas dapat dipindahkan secara langsung dari tiang A ke tiang B, maka piringan ke-4 dapat dengan mudah dipindahkan dari tiang A ke tiang C. Permasalahan berikutnya muncul adalah, bagaimana memindahkan 3 buah piringan dari tiang A ke tiang B. Hal ini dapat dengan mudah dilakukan jika dua buah piringan dapat dipindahkan dari tiang A ke tiang C, sehingga piringan ketiga dapat dipindahkan dari tiang A ke tiang B. Proses ini dapat dilakukan secara terus menerus hingga, hanya terdapat sebuah piringan saja yang akan dipindahkan dari satu tiang ke tiang lainnya (seperti halnya proses rekursif), seperti pada Gambar 4 berikut



.



.



.

Gambar 4. Proses Pemindahan Piringan - Towers of Hanoi dengan cara Rekursif

Berikut algoritma untuk memindahkan piringan sebanyak n, dari tiang awal ke tiang tujuan, melalui suatu tiang bantuan:

1. pindahkan piringan sebanyak $n-1$ dari tiang awal ke tiang bantuan, melalui tiang tujuan
2. pindahkan piringan ke n dari tiang awal ke tiang tujuan
3. pindahkan piringan sebanyak $n-1$ dari tiang bantuan ke tiang tujuan, melalui tiang awal

Code

Berikut code untuk penyelesaian permasalahan towers of hanoi dengan menggunakan teknik rekursif

In [0]:
```
def towers(n,awal,bantuan,tujuan):
    if n==1:
        print("Piringan - 1 dari-", awal,"ke-",tujuan)
    else:
        towers(n-1,awal,tujuan,bantuan)
        print("Piringan -",n, "dari-",awal,"ke-", tujuan)
        towers(n-1,bantuan,awal,tujuan)
```

In [0]:
```
towers(4,'A','B','C')
```

Latihan

Buatlah visualiasi dengan menggunakan teks, perpindahan piringan dari tower asal sampai dengan tower tujuan. Visualisasikan perpindahan piringan ini satu persatu. Contoh visualisai langkah awal, dapat dilihat pada Gambar berikut ini,

```
Pemindahan  4 lempengan dari A ke C dengan menggunakan bantuan B
A:
|1|
|2|
|3|
|4|
B:
C:
Lempengan - 1 dari- A ke- B
A:
|2|
|3|
|4|
B:
|1|
C:
--
```

.

Kembali ke Menu Awal

Visualisasi Teknik Pemrograman Rekursif

Rekursif juga dapat digunakan untuk kepentingan visualisasi. Banyak gambar yang dapat dihasilkan dengan cara rekursif ini. Gambar dari pemrograman rekursif dapat dilakukan dengan bantuan modul **Turtle**. Turtle merupakan module yang dapat digunakan oleh bahasa pemrograman Python versi apapun. Module ini dapat diumpamakan seperti seekor turtle, yang dapat bergerak maju, mundur, ke kanan, ke kiri, dan sebagainya. Turtle dapat mengambar jika ekornya **down**, sehingga ekor ini akan meninggalkan jejak (gambar) ketika turtle bergerak. Turtle tidak dapat menggambar jika ekornya **up**. Ekor ini dapat diumpamakan sebagai sebuah pena, yang dapat juga diatur warna maupun ukurannya. Untuk menggunakan module turtle ini, maka harus import turtle terlebih dahulu, seperti contoh-contoh berikut.

Code

Berikut adalah pembuatan bentuk spiral secara rekursif dengan menggunakan modul Turtle

In [0]:
```
import turtle
my_turtle = turtle.Turtle()
my_win = turtle.Screen()

def draw_spiral(my_turtle, line_len):
    if line_len > 0:
        my_turtle.forward(line_len)
        my_turtle.right(90)
        draw_spiral(my_turtle, line_len - 5)

draw_spiral(my_turtle, 90)
my_win.exitonclick()
```

Fractal

Fractal merupakan cabang matematika dan memiliki banyak kesamaan dengan teknik rekursif. Fraktal memiliki bentuk dasar, dan bentuk dasar ini dikembangkan secara berulang terus menerus.

Code

Berikut adalah pembuatan bentuk fraktal sederhana, yaitu pohon, secara rekursif dengan menggunakan modul Turtle

In [1]:
```
import turtle
def tree(branch_len, t):
    t.speed('slowest')
    if branch_len > 5:
        t.forward(branch_len)
        t.right(20)
        tree(branch_len - 15, t)
        t.left(40)
        tree(branch_len - 15, t)
        t.right(20)
        t.backward(branch_len)

def main():
    my_win = turtle.Screen()
    t = turtle.Turtle()

    t.shape("turtle")
    t.left(90)
    t.up()
    t.backward(100)
    t.down()
    t.color("green")
    tree(60, t)
    my_win.exitonclick()
main()
```

```
---------------------------------------------------------------------------
TclError                                  Traceback (most recent call last)
<ipython-input-1-ae34a7335066> in <module>()
     23     tree(60, t)
     24     my_win.exitonclick()
---> 25 main()

<ipython-input-1-ae34a7335066> in main()
     12
     13 def main():
---> 14     my_win = turtle.Screen()
     15     t = turtle.Turtle()
     16

/usr/lib/python3.6/turtle.py in Screen()
   3660     else return the existing one."""
   3661     if Turtle._screen is None:
-> 3662         Turtle._screen = _Screen()
   3663     return Turtle._screen
   3664

/usr/lib/python3.6/turtle.py in __init__(self)
   3676         # preserved (perhaps by passing it as an optional parameter)
   3677         if _Screen._root is None:
-> 3678             _Screen._root = self._root = _Root()
   3679             self._root.title(_Screen._title)
   3680             self._root.ondestroy(self._destroy)

/usr/lib/python3.6/turtle.py in __init__(self)
    432     """Root class for Screen based on Tkinter."""
    433     def __init__(self):
--> 434         TK.Tk.__init__(self)
    435
    436     def setupcanvas(self, width, height, cwidth, cheight):

/usr/lib/python3.6/tkinter/__init__.py in __init__(self, screenName, baseNam
e, className, useTk, sync, use)
   2018                 baseName = baseName + ext
   2019         interactive = 0
-> 2020         self.tk = _tkinter.create(screenName, baseName, className, in
teractive, wantobjects, useTk, sync, use)
   2021         if useTk:
   2022             self._loadtk()

TclError: no display name and no $DISPLAY environment variable
```

Segitiga Sierpinski

Segitia Sierpinski ini juga merupakan bentuk fraktal. Segitiga ini memiliki bentuk dasar (yaitu, suatu segitiga dan didalamnya terdapat empat buat segitiga dengan ukuran yang sama), dan bentuk dasar ini terus menerus dibangkitkan sampai derajat tertentu.

Code

Berikut contoh code untuk pembuatan Segitiga Sierpinski

In [0]:
```
import turtle
def draw_triangle(points, color, my_turtle):
    my_turtle.speed('slowest')
    my_turtle.fillcolor(color)
    my_turtle.up()
    my_turtle.goto(points[0][0],points[0][1])
    my_turtle.down()
    my_turtle.begin_fill()
    my_turtle.goto(points[1][0], points[1][1])
    my_turtle.goto(points[2][0], points[2][1])
    my_turtle.goto(points[0][0], points[0][1])
    my_turtle.end_fill()
def get_mid(p1, p2):
    return ((p1[0] + p2[0]) / 2, (p1[1] + p2[1]) / 2)
def sierpinski(points, degree, my_turtle):
    color_map = ["blue", "red" , "green", "white" , "yellow" ,"violet" , "oran
ge"]
    draw_triangle(points, color_map[degree], my_turtle)
    if degree > 0:
        sierpinski([points[0],get_mid(points[0], points[1]), get_mid(points[0
], points[2])], degree-1, my_turtle)
        sierpinski([points[1],get_mid(points[0], points[1]), get_mid(points[1
], points[2])], degree-1, my_turtle)
        sierpinski([points[2],get_mid(points[2], points[1]), get_mid(points[0
], points[2])], degree-1, my_turtle)

def main():
    my_turtle = turtle.Turtle()
    my_win = turtle.Screen()
    my_points = [[-100, -50], [0, 100], [100, -50]]
    sierpinski(my_points, 2, my_turtle)
    my_win.exitonclick()

main()
```

Kembali ke Menu Awal