The Migration function

The general form of the logistic function

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}, where$$

L - the maximum (in our case 1)

k - the steepness of the curve (the sign flips it, more below)

x0 - the midpoint of the curve - if the range is symmetrical towards 0 then it's dropped

This is a coded and plotted version:

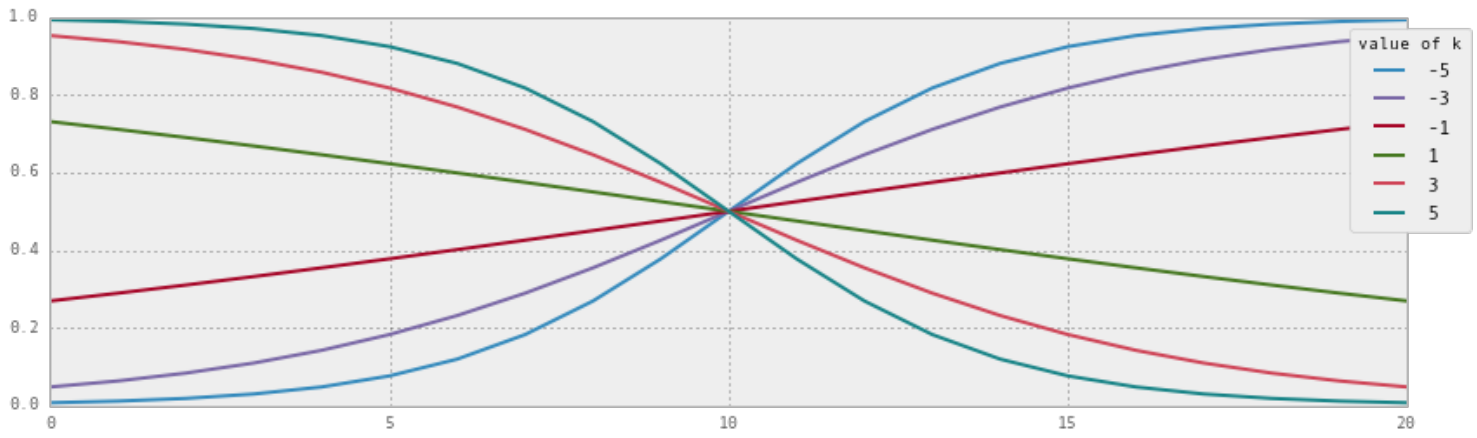
In [2]:

```
from __future__ import division

import math
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
pd.options.display.mpl_style = 'default'
%matplotlib inline
```

```
/Users/iar1g09/anaconda/lib/python2.7/site-packages/matplotlib/__i
nit__.py:892: UserWarning: axes.color_cycle is deprecated and repl
aced with axes.prop_cycle; please use the latter.
  warnings.warn(self.msg_depr % (key, alt_key))
/Users/iar1g09/anaconda/lib/python2.7/site-packages/matplotlib/__i
nit__.py:872: UserWarning: axes.color_cycle is deprecated and repl
aced with axes.prop_cycle; please use the latter.
  warnings.warn(self.msg_depr % (key, alt_key))
```

In [3]:

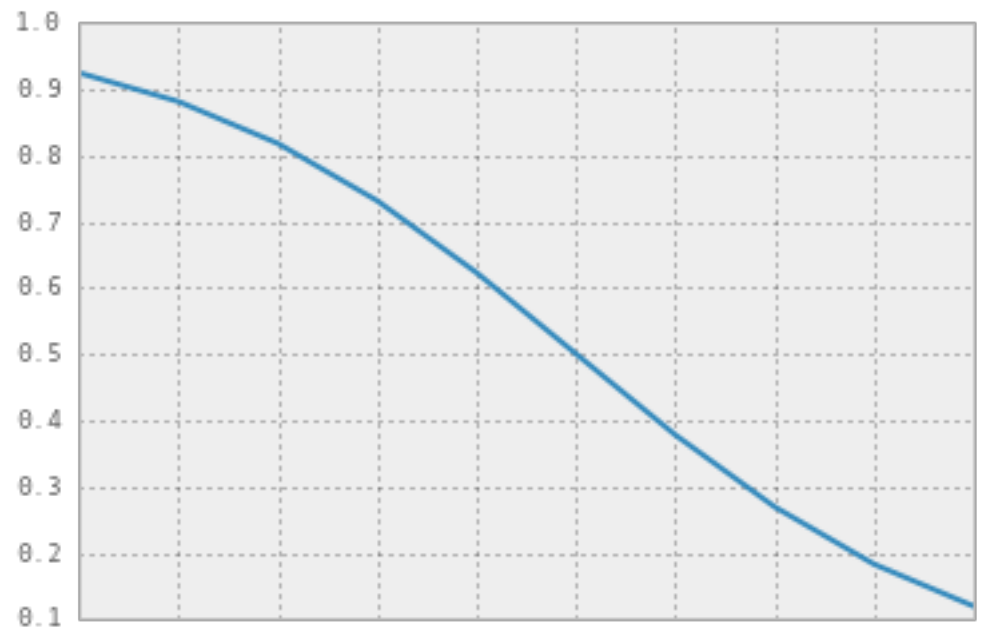
```
for k in range(-5, 6, 2):

    log_fig = plt.figure(1, figsize = (14, 4))
    plt.plot(map(lambda x : 1 / (1 + math.e**(k*x)), [x / 10.0 for x in range(
-10, 11, 1)]), label = k, linewidth = 2)

    plt.legend(bbox_to_anchor=(0.95, 1), loc=2, title = "value of k")
#plt.tick_params(labelbottom='off')
```



Plugging the fitness values - range between 0-1, k is set to 5

In [4]:

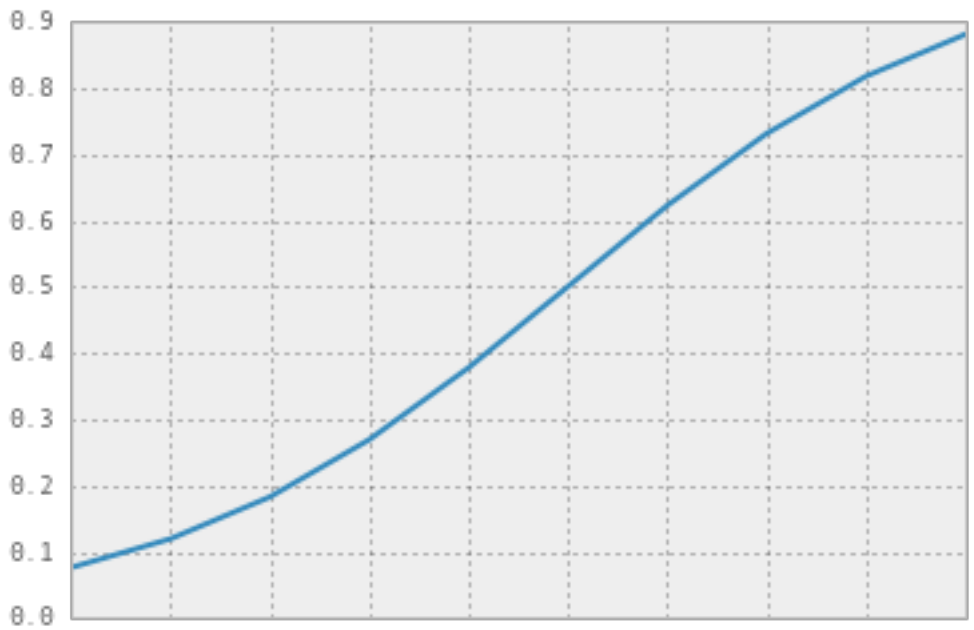
```
k = 5
plt.plot(map(lambda x : 1 / (1 + math.e**((k)*(x-0.5))), [x / 10.0 for x in ra
nge(0, 10, 1)]), label = x, linewidth = 2)
plt.tick_params(labelbottom='off')
```



The lower the fitness the higher the chance of migrating. We can also swap it over by changing the sign of k.

In [5]:

```
k = -5
plt.plot(map(lambda x : 1 / (1 + math.e**((k)*(x-0.5))), [x / 10.0 for x in ra
nge(0, 10, 1)]), label = x, linewidth = 2)
plt.tick_params(labelbottom='off')
```



Population density

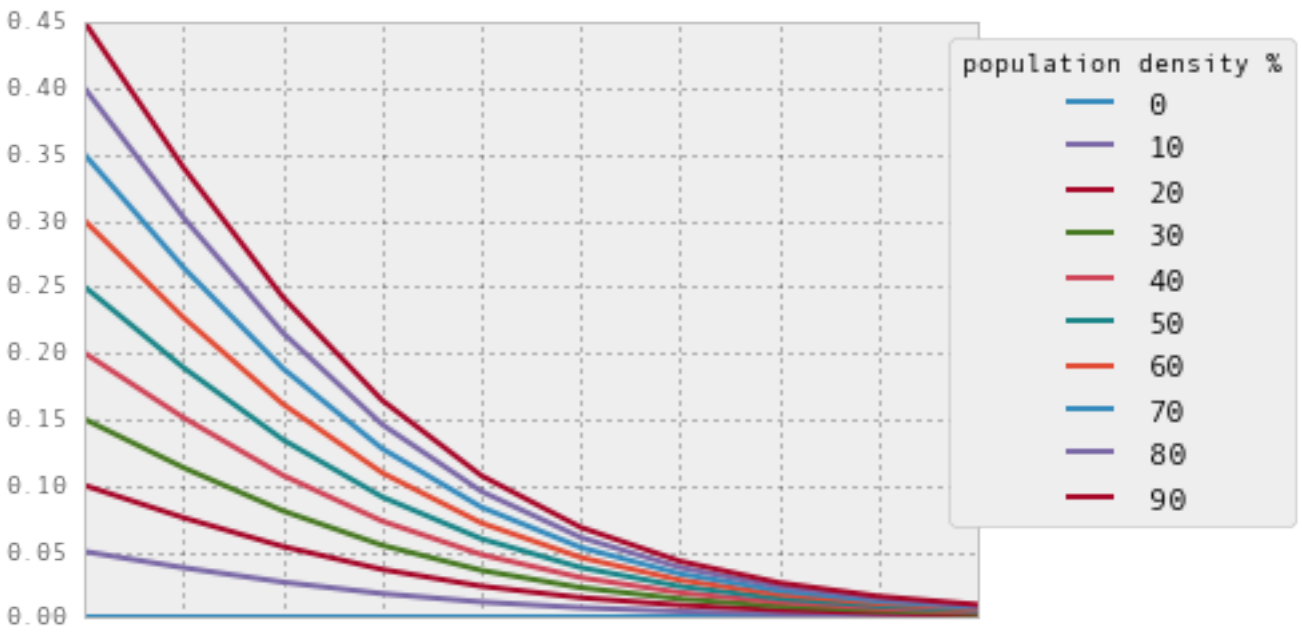
Now adding the population density component. The more people the higher chance of migrating in general.

In [6]:

```
cc = 100
k = 5
for people in range(0, 100, 10):
    plt.plot(map(lambda x : (people/ cc) * (1 / (1 + math.e**(k*x))), [x / 10.
0 for x in range(0, 10, 1)]), label = people, linewidth = 2)
plt.tick_params(labelbottom='off')
plt.legend(bbox_to_anchor=(0.95, 1), loc=2, title = "population density %")
```

Out[6]:

```
<matplotlib.legend.Legend at 0x10a705dd0>
```



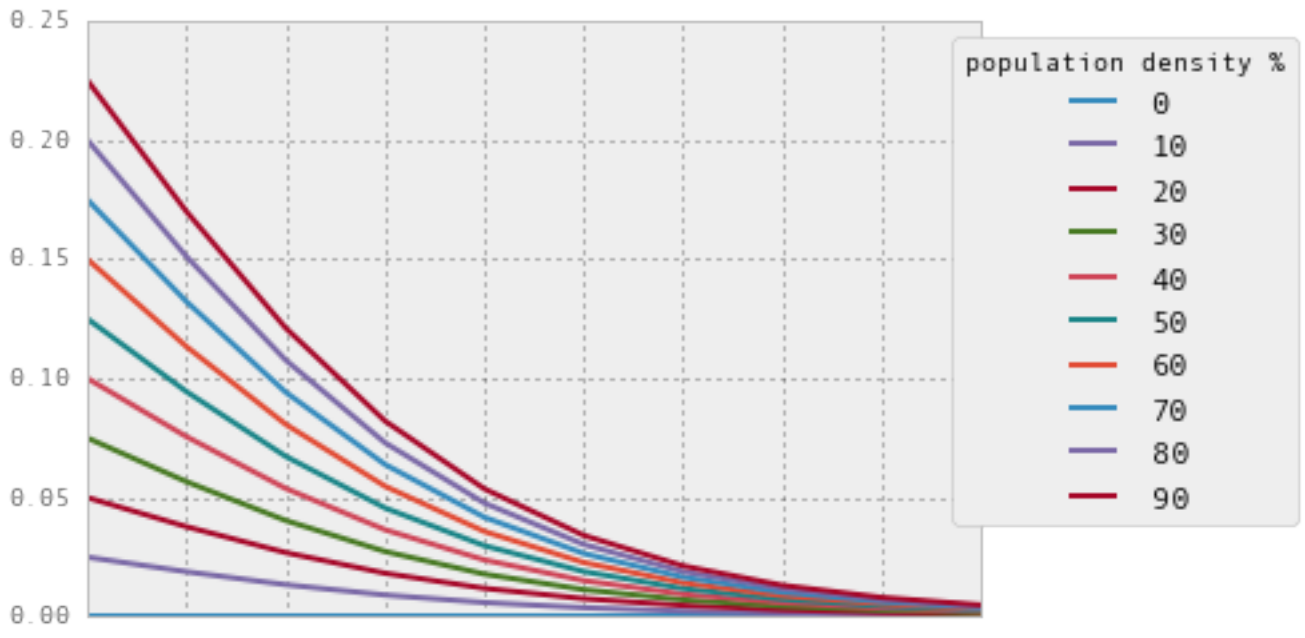
An additional term can be used for experiments in which we vary the migration rate (the y-axis shows the difference).

In [7]:

```
cc = 100
k = 5
mig_const = 0.5 #halving the migration rate
for people in range(0, 100, 10):
    plt.plot(map(lambda x : mig_const * (people/ cc) * (1 / (1 + math.e**(k*x)
)), [x / 10.0 for x in range(0, 10, 1)]), label = people, linewidth = 2)
plt.tick_params(labelbottom='off')
plt.legend(bbox_to_anchor=(0.95, 1), loc=2, title = "population density %")
```

Out[7]:

<matplotlib.legend.Legend at 0x1087daf50>



Considering individual fitness in respect to the average fitness

$$x = \frac{f_i - \bar{f}}{\bar{f}}, where$$

f(i) - fitness of an individual i

f dash - average fitness in the population

with a positive value for the k parameter (e.g., k=5):

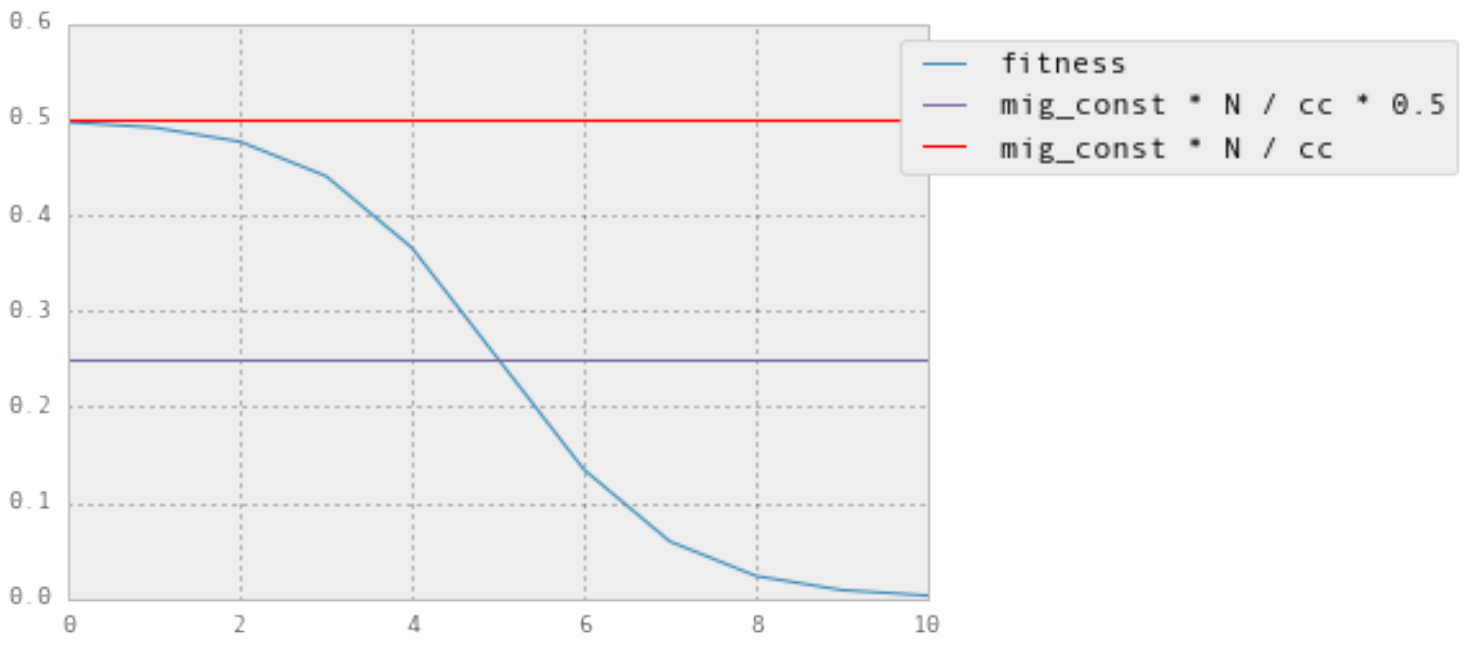
- very negative x values (i.e., below average fitness) will give migration probabilities that get close to m x N/C
- very positive x values (i.e., above average fitness) will give migration probabilities that get close to 0 (but never get to zero)
- zero x values (i.e., exactly average fitness) will give migration probabilities of m x N/C x 0.5

In [8]:

```
people = 50
cc = 100
k = 5

# test 10 values of fitness from 0.0 to 1.0 every 0.1
fitnesses = [x / 10.0 for x in range(0, 11, 1)]


# mean fitness  (= 0.5)
m_fitness = reduce(lambda x, y: x + y, fitnesses) / len(fitnesses)
# migration constant of 1
mig_const = 1
a = []
b = []
c = []

for fitness_value in fitnesses:
    x = (fitness_value - m_fitness) / m_fitness
    a.append(mig_const * (people/ cc) * (1 / (1 + math.e**(k*(x)))))
    b.append(mig_const * (people / cc) * 0.5)
    c.append(mig_const * (people / cc))

plt.plot(a, label = 'fitness')
plt.plot(b, label = 'mig_const * N / cc * 0.5')
plt.plot(c, label = 'mig_const * N / cc', color = 'red')
plt.ylim([0.0, 0.6])
plt.legend(bbox_to_anchor=(0.95, 1), loc=2)
```

Out[8]:

```
<matplotlib.legend.Legend at 0x10aa038d0>
```



Flipped over

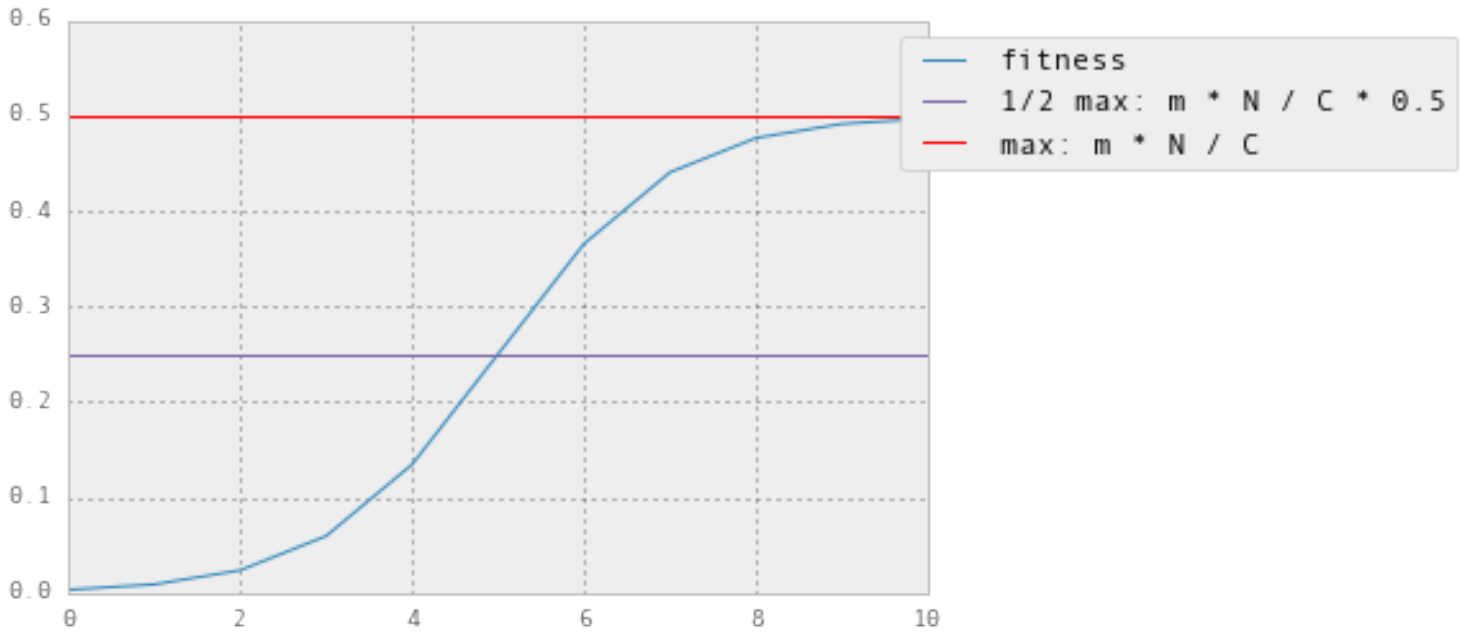
with a negative value for the k parameter e.g., k=-5)

- very negative x values (i.e., below average fitness) will give migration probabilities that get close to 0 (but never get to zero)
- very positive x values (i.e., above average fitness) will give migration probabilities that get close to m * N/C
- zero x values (i.e., exactly average fitness) will give migration probabilities of m *N/C* 1/2

In [11]:

```
people = 50
cc = 100
k = -5

# test 10 values of fitness from 0.0 to 1.0 every 0.1
fitnesses = [x / 10.0 for x in range(0, 11, 1)]

# mean fitness  (= 0.5)
m_fitness = reduce(lambda x, y: x + y, fitnesses) / len(fitnesses)

# migration constant of 1
mig_const = 1
a = []
b = []
c = []

for fitness_value in fitnesses:
    x = (fitness_value - m_fitness) / m_fitness
    a.append(mig_const * (people / cc) * (1 / (1 + math.e**(k*(x)))))
    b.append(mig_const * (people / cc) * 0.5)
    c.append(mig_const * (people / cc))

plt.plot(a, label = 'fitness')
plt.plot(b, label = '1/2 max: m * N / C * 0.5')
plt.plot(c, label = 'max: m * N / C', color = 'red')
plt.ylim([0.0, 0.6])
plt.legend(bbox_to_anchor=(0.95, 1), loc=2)
```

```
Out[11]:

<matplotlib.legend.Legend at 0x10b0a1550>
```



Finally for k = 0

with k set to 0

- all individuals will have an "average" migration probability = m x N/C x 0.5 irrespective of their fitness

In [63]:

```
people = 50
cc = 100
k = 0

# test 10 values of fitness from 0.0 to 1.0 every 0.1
fitnesses = [x / 10.0 for x in range(0, 11, 1)]

# mean fitness  (= 0.5)
m_fitness = reduce(lambda x, y: x + y, fitnesses) / len(fitnesses)

# migration constant of 0.1
mig_const = 1
a = []
b = []
c = []

for fitness_value in fitnesses:
    x = (fitness_value - m_fitness) / m_fitness
    a.append(mig_const * (people / cc) * (1 / (1 + math.e**(k*(x)))))
    b.append(mig_const * (people / cc) * 0.5)
    c.append(mig_const * (people / cc))

plt.plot(a, label = 'fitness', linewidth = 2)
plt.plot(b, label = 'm * N / C * 0.5', color = 'red')
plt.plot(c, label = 'm * N / C')
plt.ylim([0.0, 0.6])
plt.legend(bbox_to_anchor=(0.95, 1), loc=2)
```

Out[63]:

```
<matplotlib.legend.Legend at 0x13d61ef0>
```