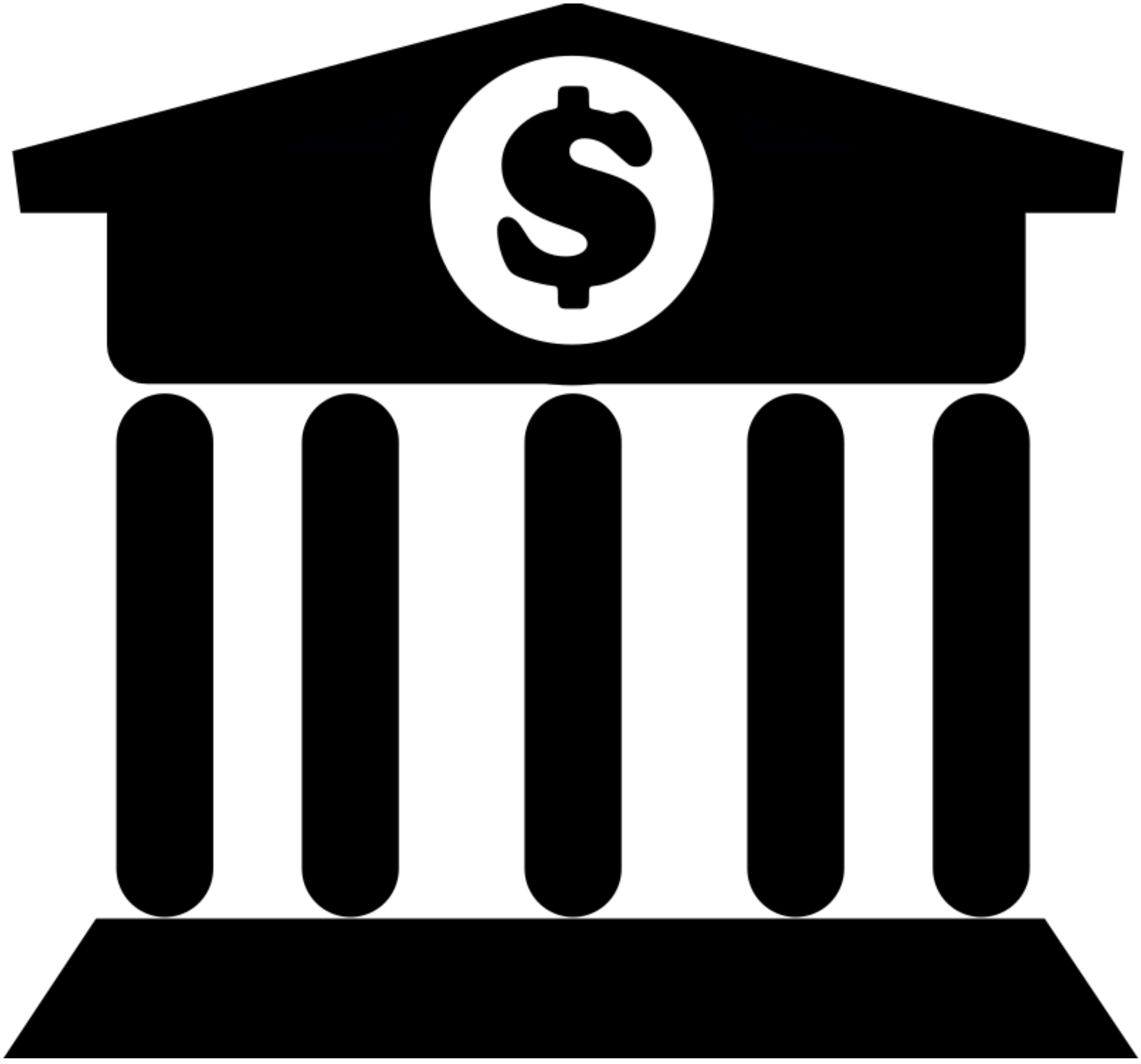
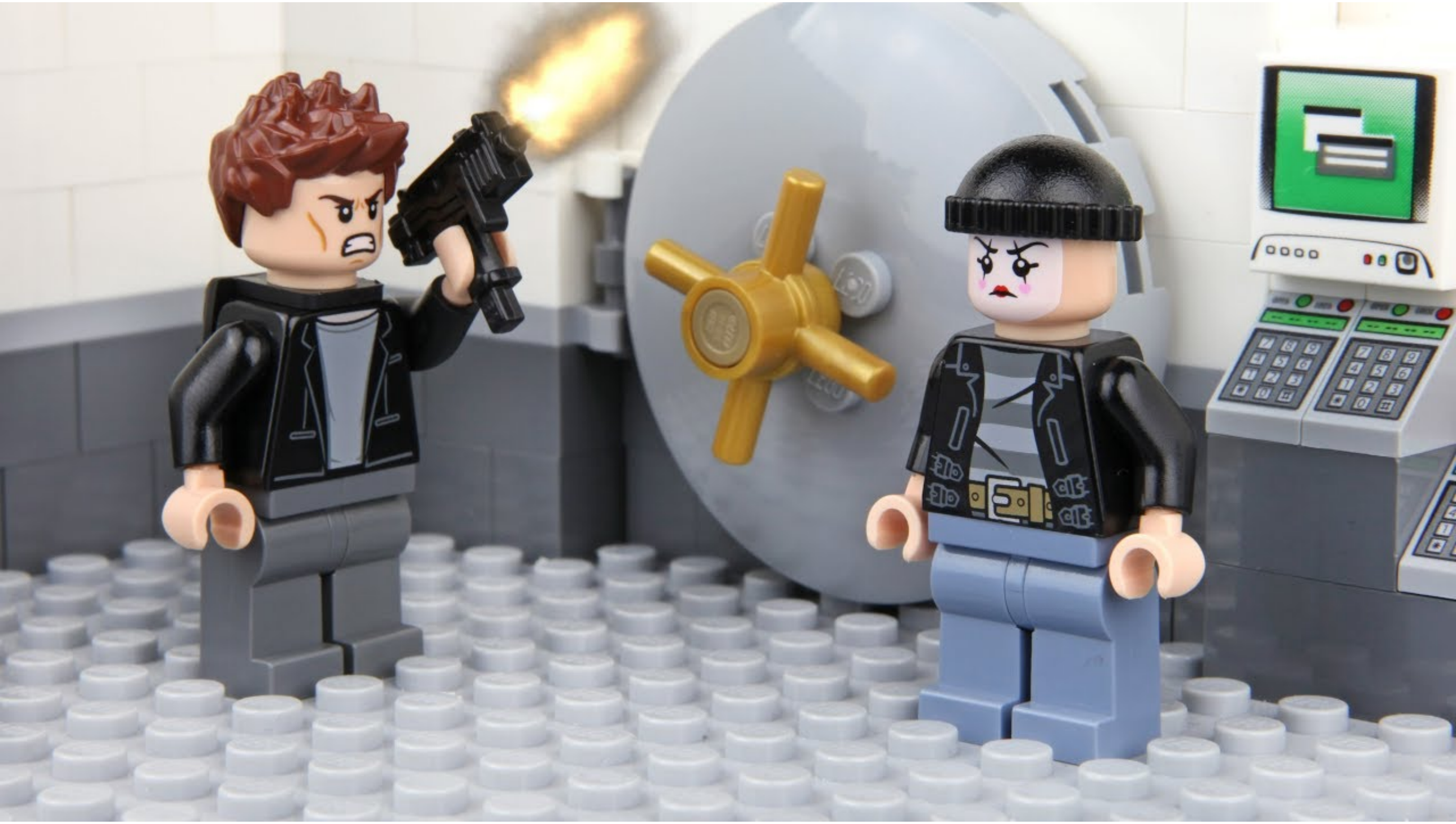
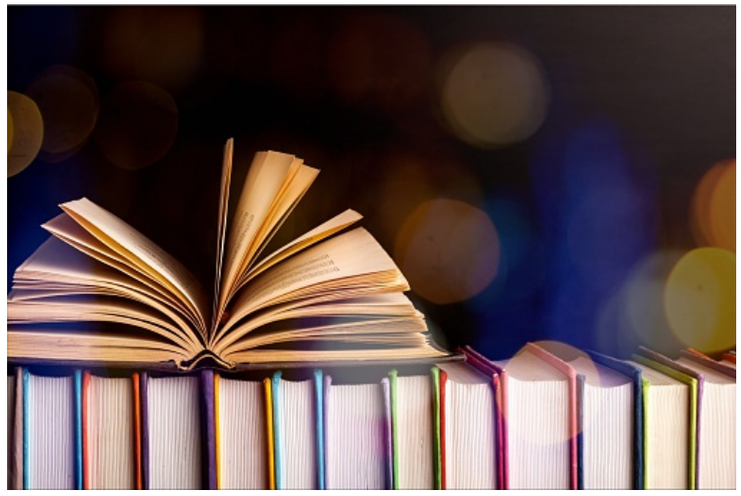
The Knapsack Problem

Use it vs Lose it

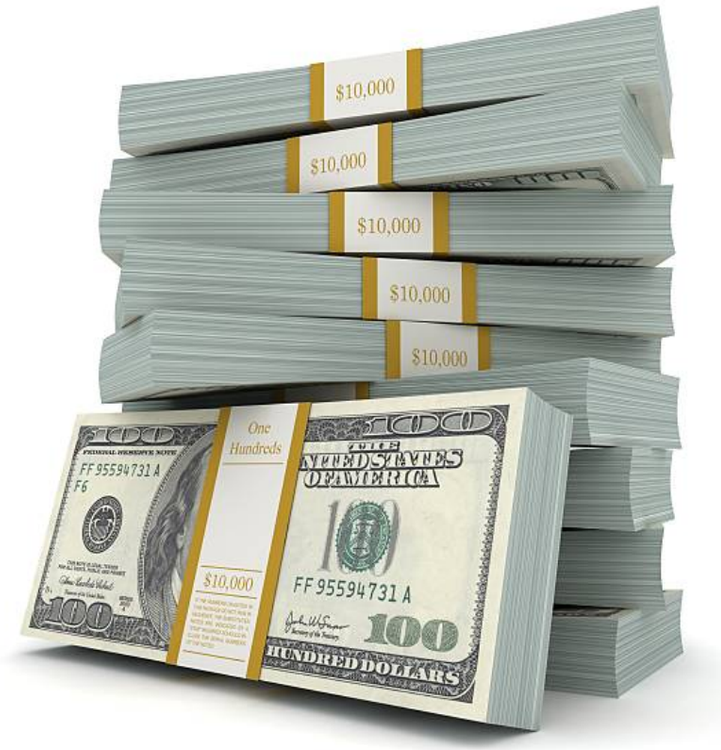




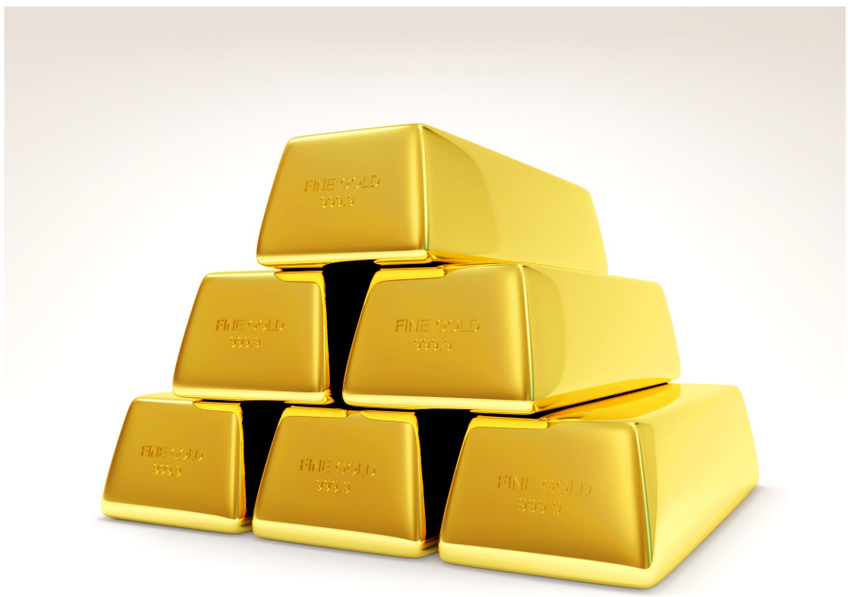
Diamonds



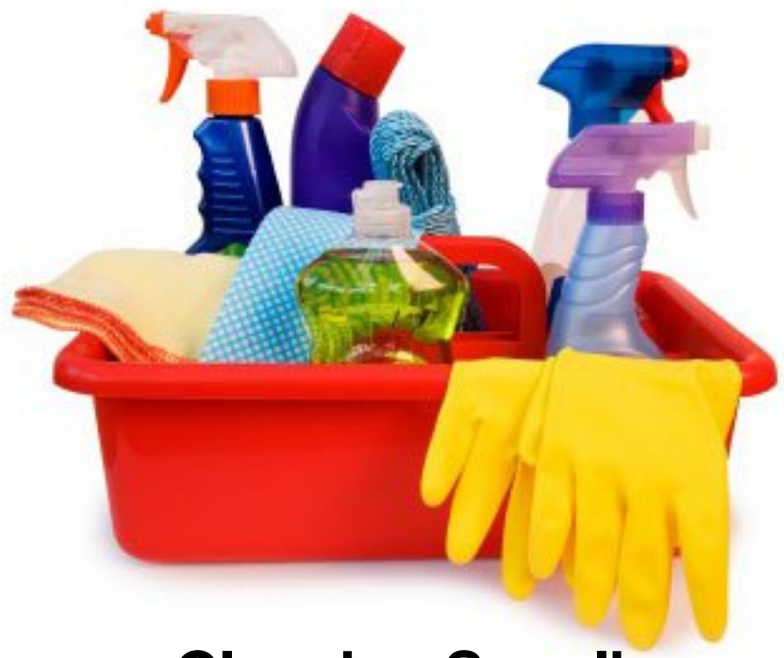
Books



Dollars

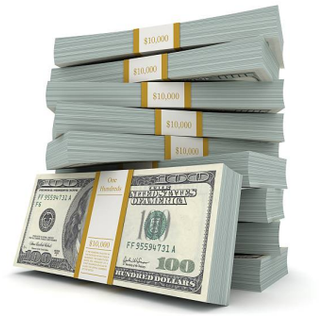


Gold



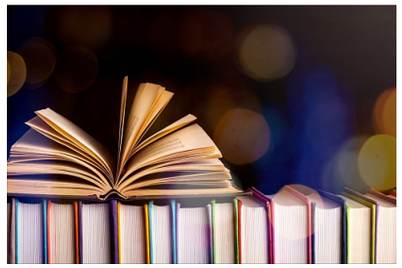
Cleaning Supplies

Gold

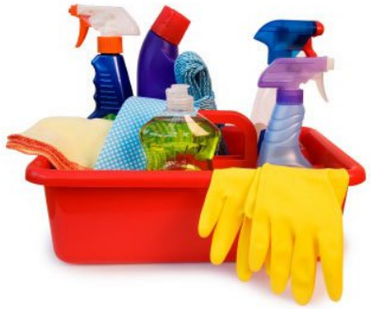


Dollars

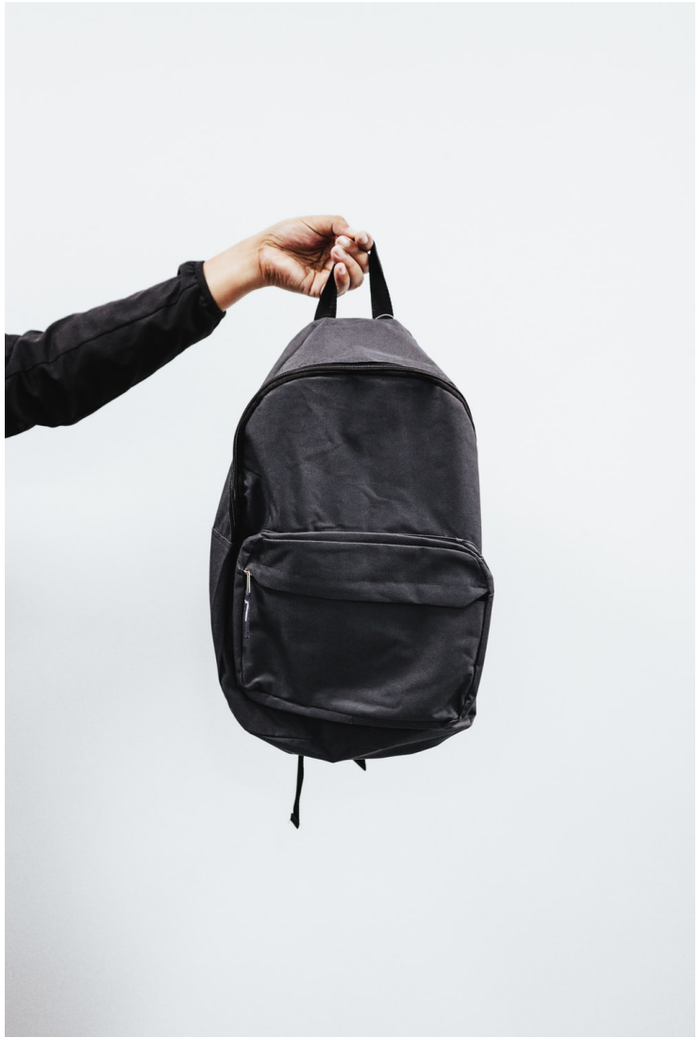
Diamonds



Books

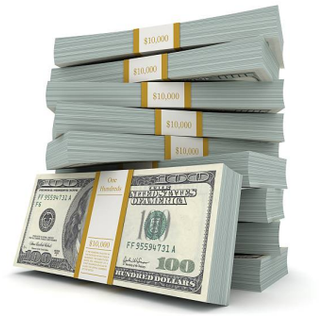


Cleaning Supplies



Gold

19 lbs

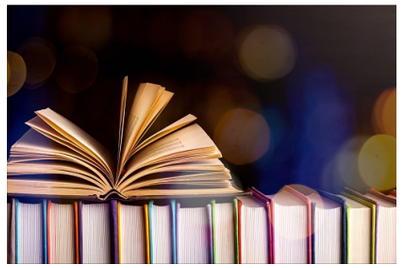


Dollars

12 lbs

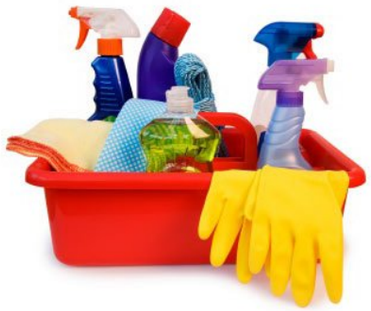
Diamonds

9 lbs



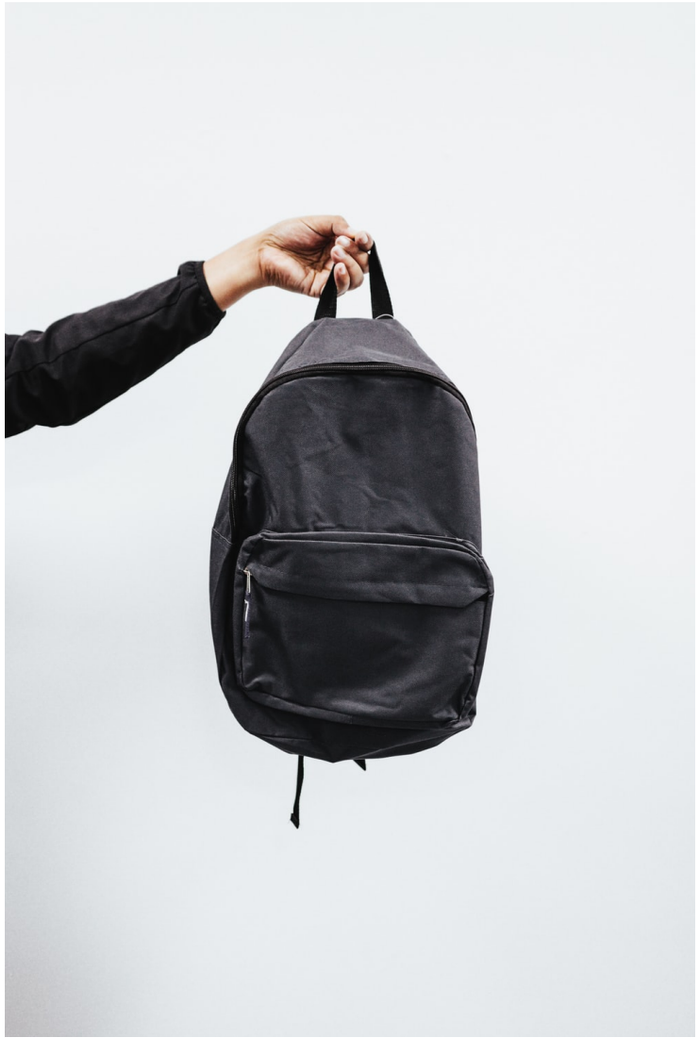
Books

12 lbs



Cleaning Supplies

5 lbs

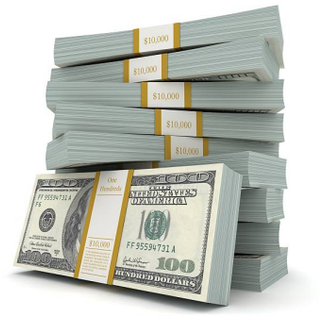


Capacity 20 lbs

Gold

19 lbs

200K $



Dollars

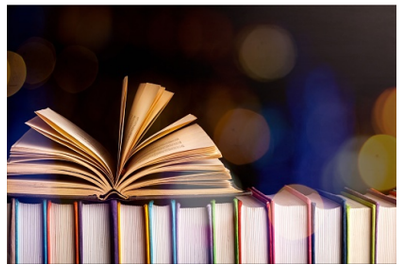
12 lbs

100K $

Diamonds

9 lbs

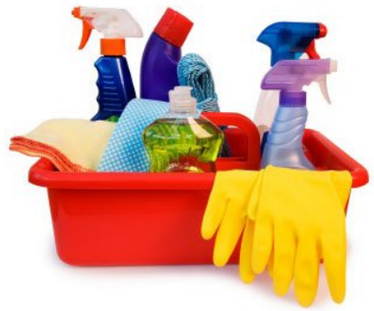
1.5M $



Books

12 lbs

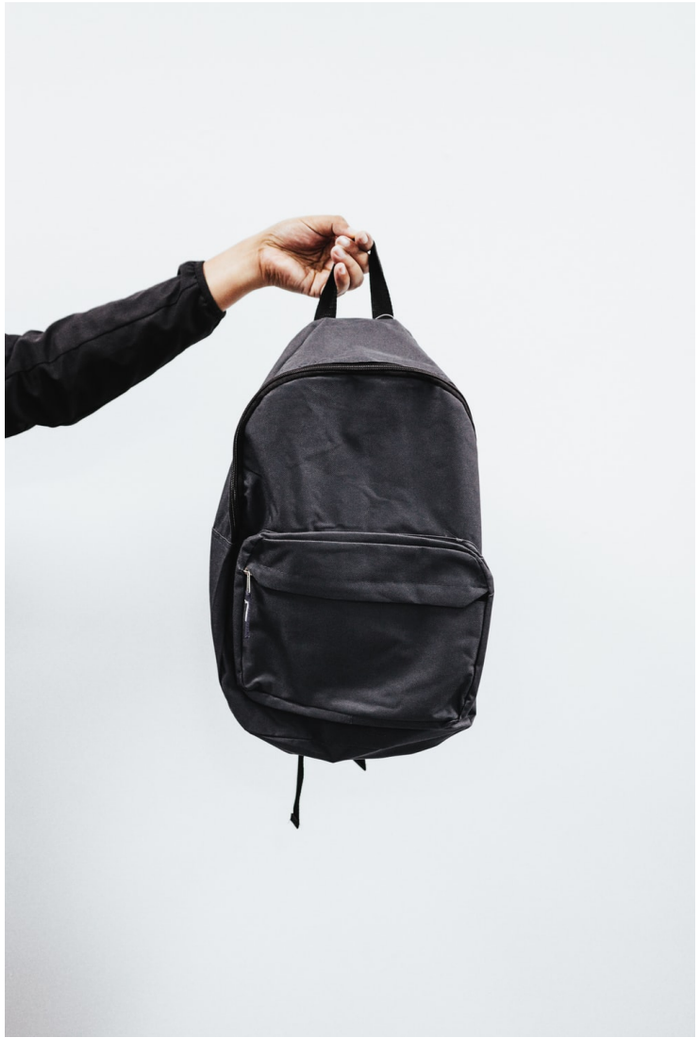
700 $



Cleaning Supplies

5 lbs

12 $

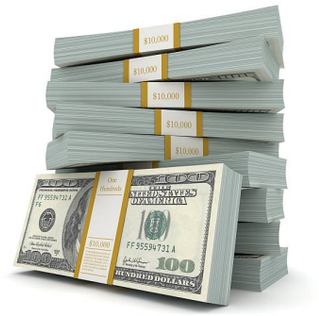


Capacity 20 lbs

Gold

19 lbs

200K $



Dollars

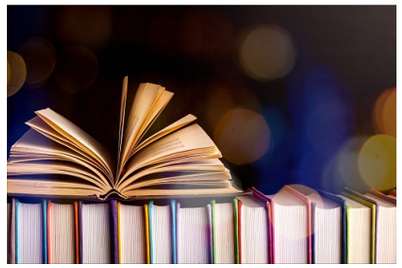
12 lbs

100K $

Diamonds

9 lbs

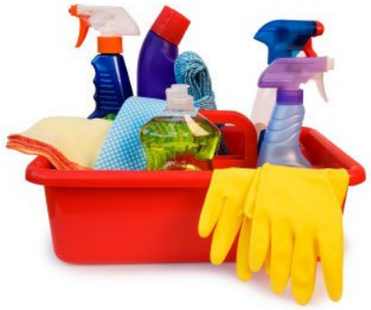
1.5M $



Books

12 lbs

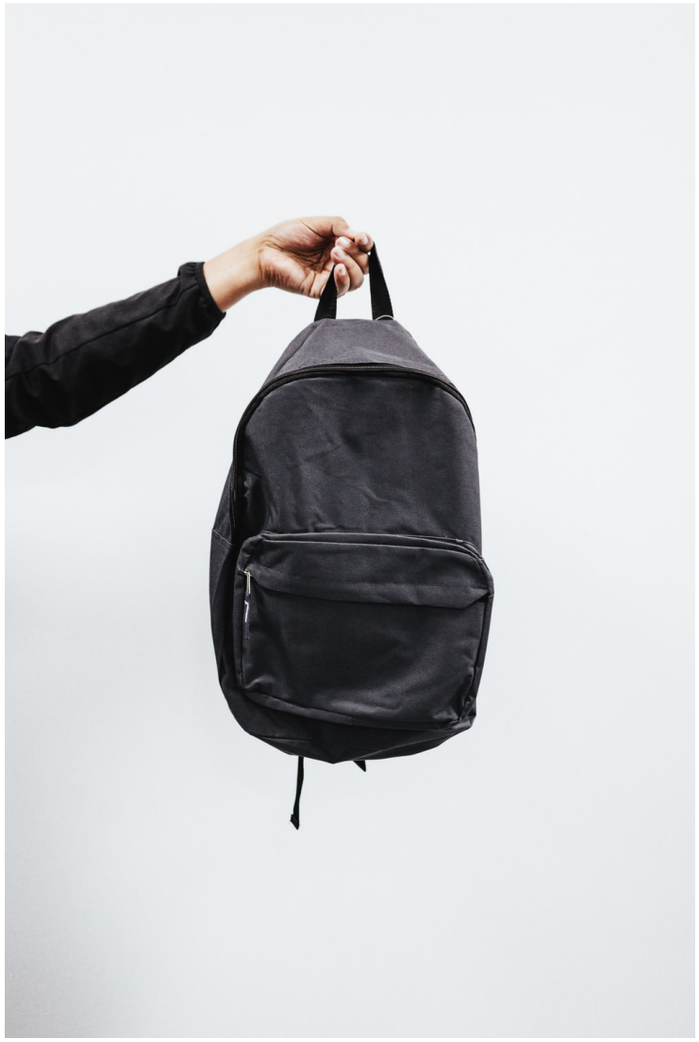
700 $



Cleaning Supplies

5 lbs

12 $



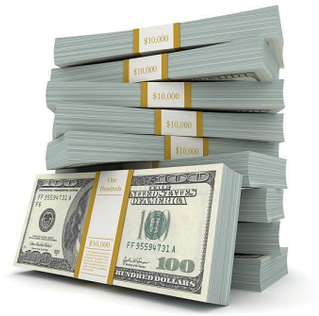
Capacity 20 lbs

You also have N items that you might want to take with you back home. Unfortunately you can not fit all of them in the knapsack so you will have to choose. For each item you are given its weight and its value. You want to maximize the total value of all the items you are going to bring. What is this maximum total value?

Gold

19 lbs

200K $



Dollars

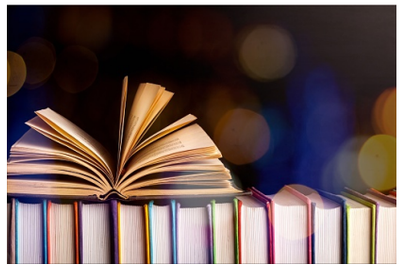
12 lbs

100K $

Diamonds

9 lbs

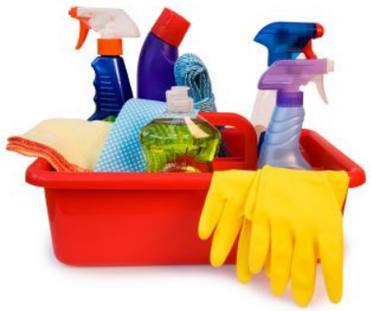
1.5M $



Books

12 lbs

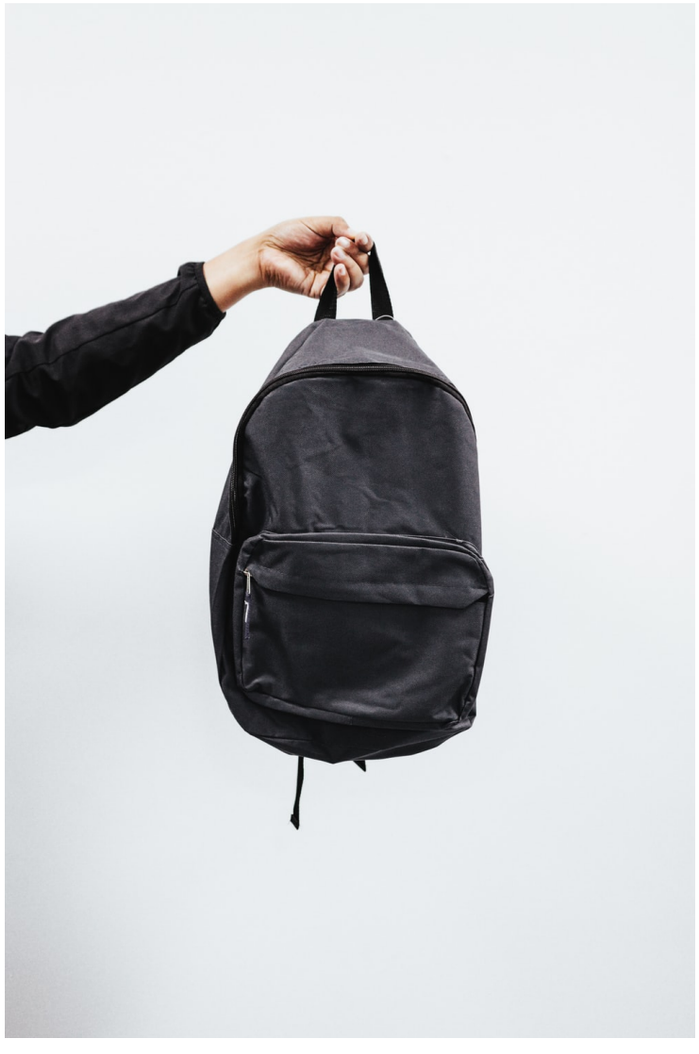
700 $



Cleaning Supplies

5 lbs

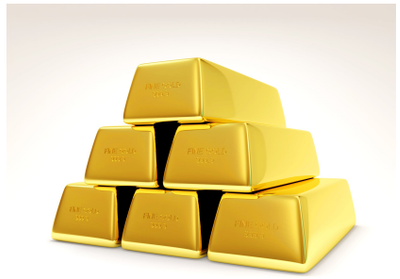
12 $



Capacity 20 lbs

```
def knapsack(capacity, items):
    #TODO
```

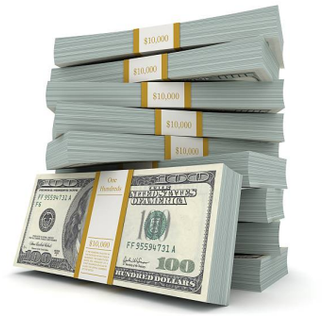
knapsack(20, [[19, 2*105], [12, 10**5], [9, 1.5*10**6], [12, 700], [5, 12]])**



Gold

19 lbs

200K $



Dollars

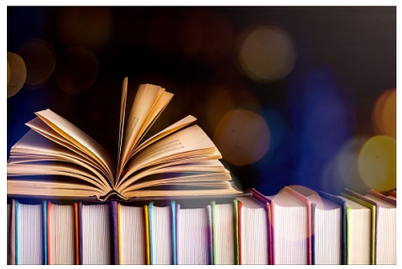
12 lbs

100K $

Diamonds

9 lbs

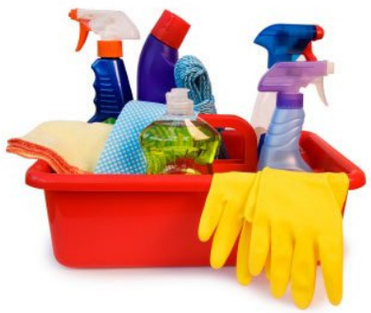
1.5M $



Books

12 lbs

700 $



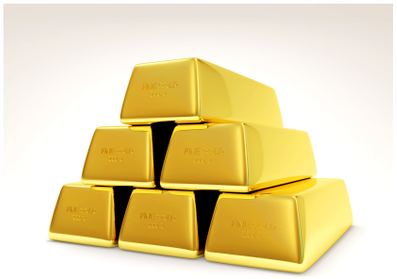
Cleaning Supplies

5 lbs

12 $

```
def knapsack(capacity, items):
        if len(items) == 0 or capacity <= 0:    #BASE CASES
            return 0
```

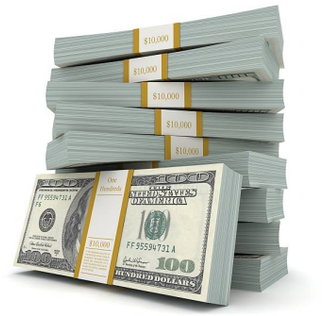
knapsack(20, [[19, 2*105], [12, 10**5], [9, 1.5*10**6], [12, 700], [5, 12]])**



Gold

19 lbs

200K $



Dollars

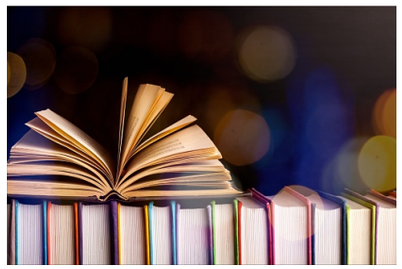
12 lbs

100K $

Diamonds

9 lbs

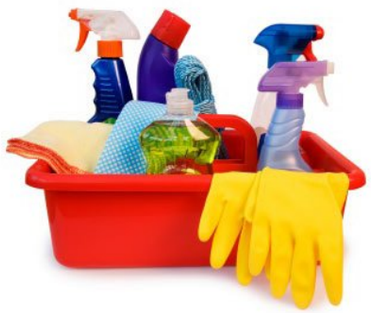
1.5M $



Books

12 lbs

700 $



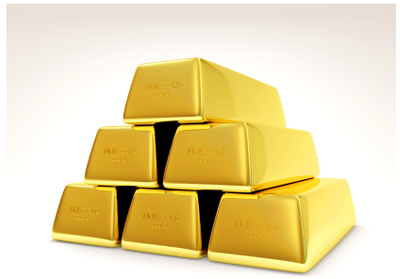
Cleaning Supplies

5 lbs

12 $

```
def knapsack(capacity, items):
        if len(items) == 0 or capacity <= 0:    #BASE CASES
            return 0
    elif items[0][0] > capacity: #first item does not fit in the backpack
        return knapsack(capacity, items[1:])
```

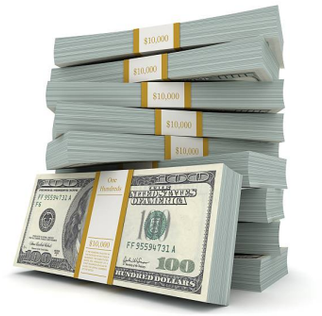
knapsack(20, [[19, 2*105], [12, 10**5], [9, 1.5*10**6], [12, 700], [5, 12]])**



Gold

19 lbs

200K $



Dollars

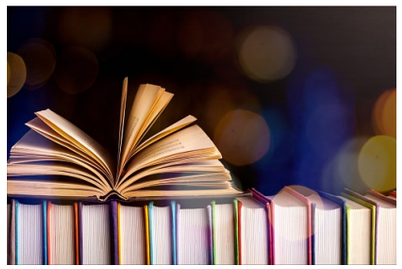
12 lbs

100K $

Diamonds

9 lbs

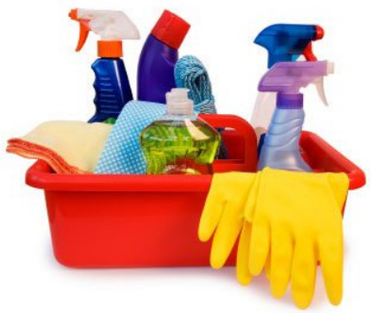
1.5M $



Books

12 lbs

700 $



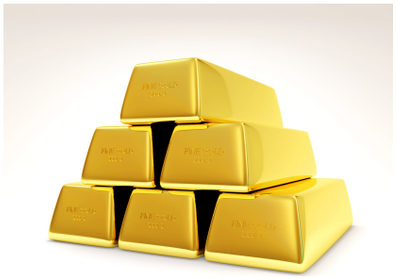
Cleaning Supplies

5 lbs

12 $

```
def knapsack(capacity, items):
        if len(items) == 0 or capacity <= 0:    #BASE CASES
            return 0
    elif items[0][0] > capacity: #first item does not fit in the backpack
          return knapsack(capacity, items[1:])
     else:
        use_it = items[0][1]+ knapsack(capacity - items[0][0], items[1:]) #Keep the item
```

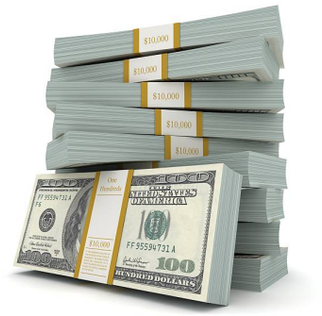
knapsack(20, [[19, 2*105], [12, 10**5], [9, 1.5*10**6], [12, 700], [5, 12]])**



Gold

19 lbs

200K $



Dollars

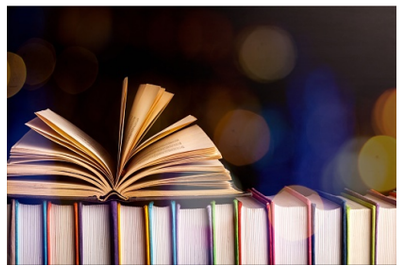
12 lbs

100K $

Diamonds

9 lbs

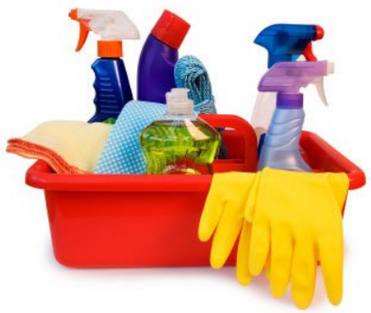
1.5M $



Books

12 lbs

700 $



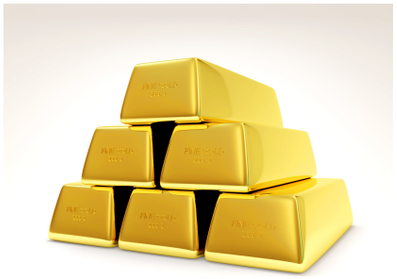
Cleaning Supplies

5 lbs

12 $

```
def knapsack(capacity, items):
        if len(items) == 0 or capacity <= 0:    #BASE CASES
            return 0
    elif items[0][0] > capacity: #first item does not fit in the backpack
          return knapsack(capacity, items[1:])
     else:
        use_it = items[0][1]+ knapsack(capacity - items[0][0], items[1:]) #Keep the item
        lose_it = knapsack(capacity, items[1:]) #Drop the item
```

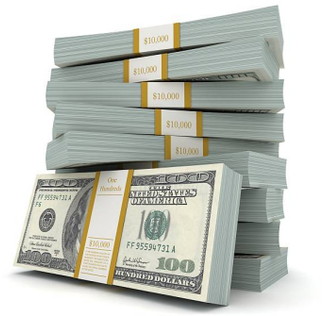
knapsack(20, [[19, 2*105], [12, 10**5], [9, 1.5*10**6], [12, 700], [5, 12]])**



Gold

19 lbs

200K $



Dollars

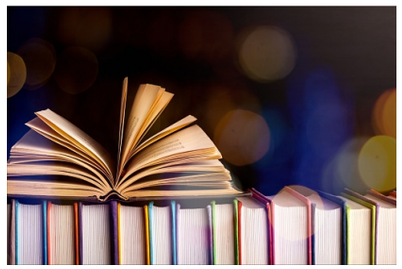
12 lbs

100K $

Diamonds

9 lbs

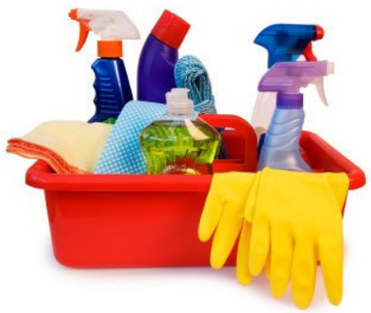
1.5M $



Books

12 lbs

700 $



Cleaning Supplies

5 lbs

12 $

```
def knapsack(capacity, items):
        if len(items) == 0 or capacity <= 0:    #BASE CASES
            return 0
    elif items[0][0] > capacity: #first item does not fit in the backpack
          return knapsack(capacity, items[1:])
     else:
        use_it = items[0][1]+ knapsack(capacity - items[0][0], items[1:]) #Keep the item
        lose_it = knapsack(capacity, items[1:]) #Drop the item
        return max(use_it, lose_it) #Keep the best option
```

knapsack(20, [[19, 2*105], [12, 10**5], [9, 1.5*10**6], [12, 700], [5, 12]])**

MORGAN FREEMAN

MICHAEL CAINE

ALAN ARKIN

GOING IN STYLE