



Python Programming

Лекц 6

Багш Ж.Золжаргал
Х.Хулан

WHAT IS RECURSION?

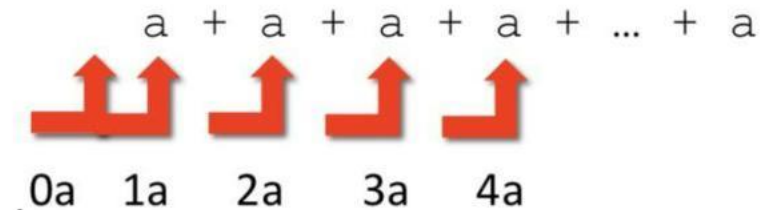


- Алгоритмаар: Хуваан захирах болон багасгаж захирах аргаар асуудлыг шийдвэрлэх
 - Асуудлыг түүний ижил жижиг асуудал болгон багасгах
- Семантикаар: Функц өөрийгөө дууддаг програмчлалын техник
 - Програмчлалд: Зорилго нь хязгааргүй давтахгүй байх
 - 1 эсвэл өөр бусад суурь тохиолдолдын шийдэл нь тодорхой байх
 - Том асуудлыг жижигрүүлэх замаар бусад оролтыг боловсруулж үр дүн гаргах

MULTIPLICATION - ITERATIVE SOLUTION



- Үржих $a * b$ нь 'a-г b удаа нэмэх' адил
- b-ээс эхэлж давтана `number(i)`
 - $i < i - 1$ ба 0 үед зогсоно
- `result` нь тухайнагшинд
 - `result <- result + a`



```
def mult_iter(a, b):  
    result = 0  
    while b > 0:  
        result += a  
        b -= 1  
    return result
```

iteration
current value of computation,
a running sum
current value of iteration variable

MULTIPLICATION - RECURSIVE SOLUTION

- Рекурсив алхам
 - Асуудлыг энгийн асуудал болгож бууруулах
- Үндсэн тохиолдол
 - Энгийн тохиолдол хүртэл асуудлыг бууруулах
 - Хүртэл $b = 1$, $a * b = a$

$$\begin{aligned} a * b &= \underbrace{a + a + a + a + \dots + a}_{b \text{ times}} \\ &= a + \underbrace{a + a + a + \dots + a}_{b-1 \text{ times}} \\ &= a + \boxed{a * (b-1)} \end{aligned}$$

recursive reduction

```
def mult(a, b):  
    if b == 1:  
        return a  
    else:  
        return a + mult(a, b-1)
```

base case

recursive step

FACTORIAL



$$n! = n * (n-1) * (n-2) * (n-3) * \dots * 1$$

- n нь хэд үед бид факториалыг нь мэдэх вэ?

$n = 1$ \rightarrow `if n == 1:`
 `return 1` *base case*

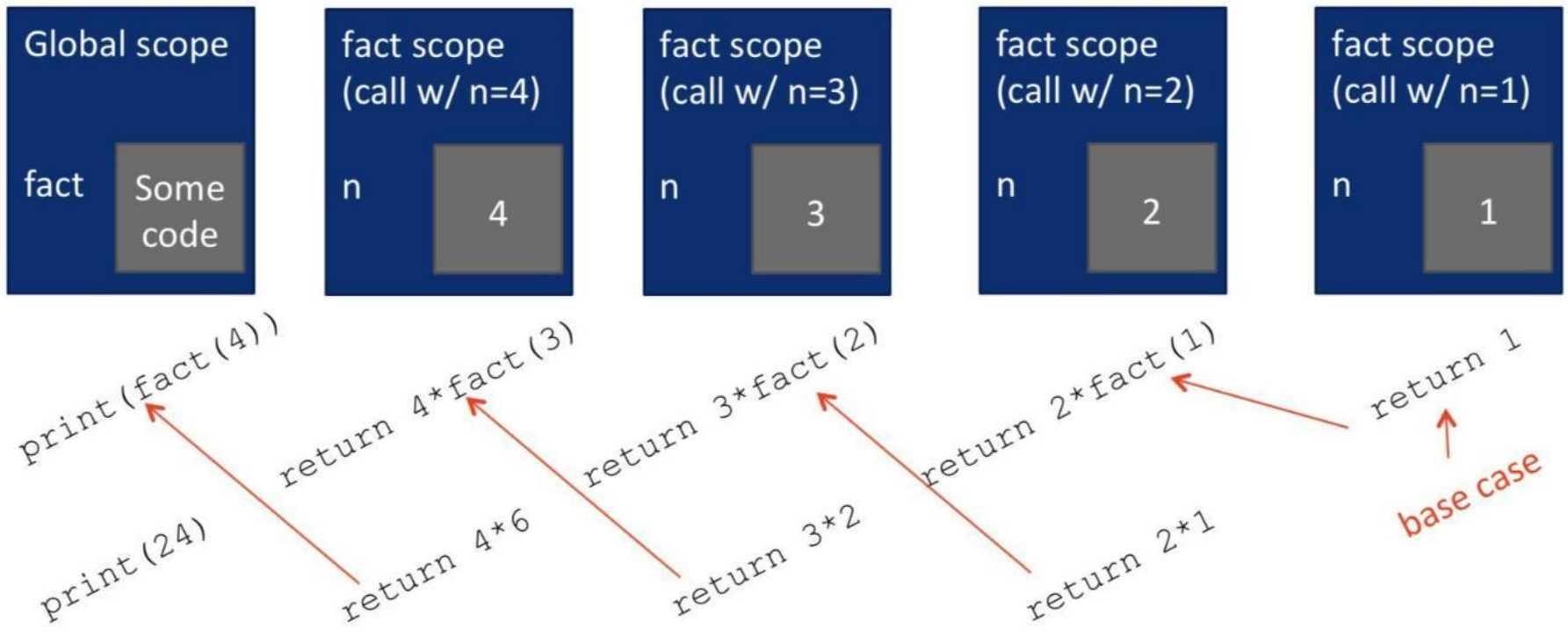
- Асуудлыг хэрхэн бууруулах вэ? Үндсэн тохиолдолд хүрэхийн тулд дахин бичнэ

$n*(n-1)!$ \rightarrow `else:`
 `return n*factorial(n-1)`

recursive step

RECURSIVE FUNCTION SCOPE EXAMPLE

```
def fact(n):  
    if n == 1:  
        return 1  
    else:  
        return n*fact(n-1)  
  
print(fact(4))
```



ITERATION

vs

RECURSION



```
def factorial_iter(n):  
    prod = 1  
    for i in range(1,n+1):  
        prod *= i  
    return prod
```

```
def factorial(n):  
    if n == 1:  
        return 1  
    else:  
        return n*factorial(n-1)
```

- Рекурсив нь энгийн
- Рекурсив нь магадгүй програмистадашигтай
- Рекурсив нь магадгүй компьютерт ашиггүй

TOWERS OF HANOI




- Түүх
 - 3 өндөр шон
 - 64 ялгаатай хэмжээтэй багц диск - нэг шонд эхэлнэ
 - Багцаар нь 2 дахь шон руу зөөх
 - Тухайн агшинд нэг л диск зөөнө, том дискийг жижиг диск дээр тавьж болохгүй

TOWERS OF HANOI



- Тухайн хөдөлгөөнүүдийг дэлгэцэнд дүрслэх програмыг хэрхэн бичих вэ?
- Рекурсивээр сэтгэ!
 - Жижиг асуудлыг шийд
 - Үндсэн асуудлыг шийд
 - Жижиг асуудлыг шийд

TOWERS OF HANOI



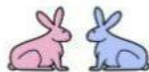
```
def printMove(fr, to):  
    print('move from ' + str(fr) + ' to ' + str(to))  
  
def Towers(n, fr, to, spare):  
    if n == 1:  
        printMove(fr, to)  
    else:  
        Towers(n-1, fr, spare, to)  
        Towers(1, fr, to, spare)  
        Towers(n-1, spare, to, fr)
```

RECURSION WITH MULTIPLE BASE CASES

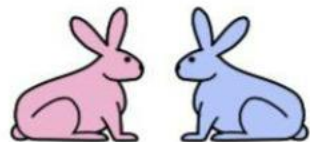


- Фибоначчийн тоонууд
 - Фибоначчи дараах сорилтыг загварчласан
 - Шинээр хос туулай төрнө (1 эр, 1 эм)
 - 1 сартайдаа өсөж том болдог
 - Туулай жирэмсэний нэг сарын хугацаатай
 - Туулай хэзээ ч үхэхгүй гэж үзье. Тэр 2 дахь сараасаа хойш сар бүр нэг шинэ хос гаргана.
 - 1 жилийн эцэст хичнээн эмэгтэй туулай байх вэ?

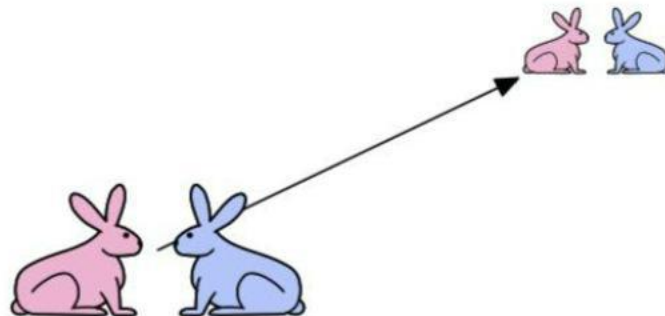
Rabbits



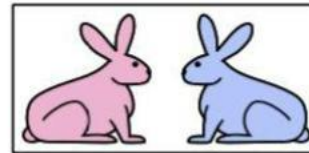
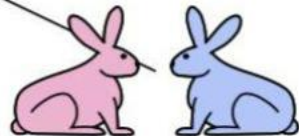
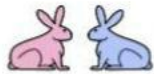
Rabbits



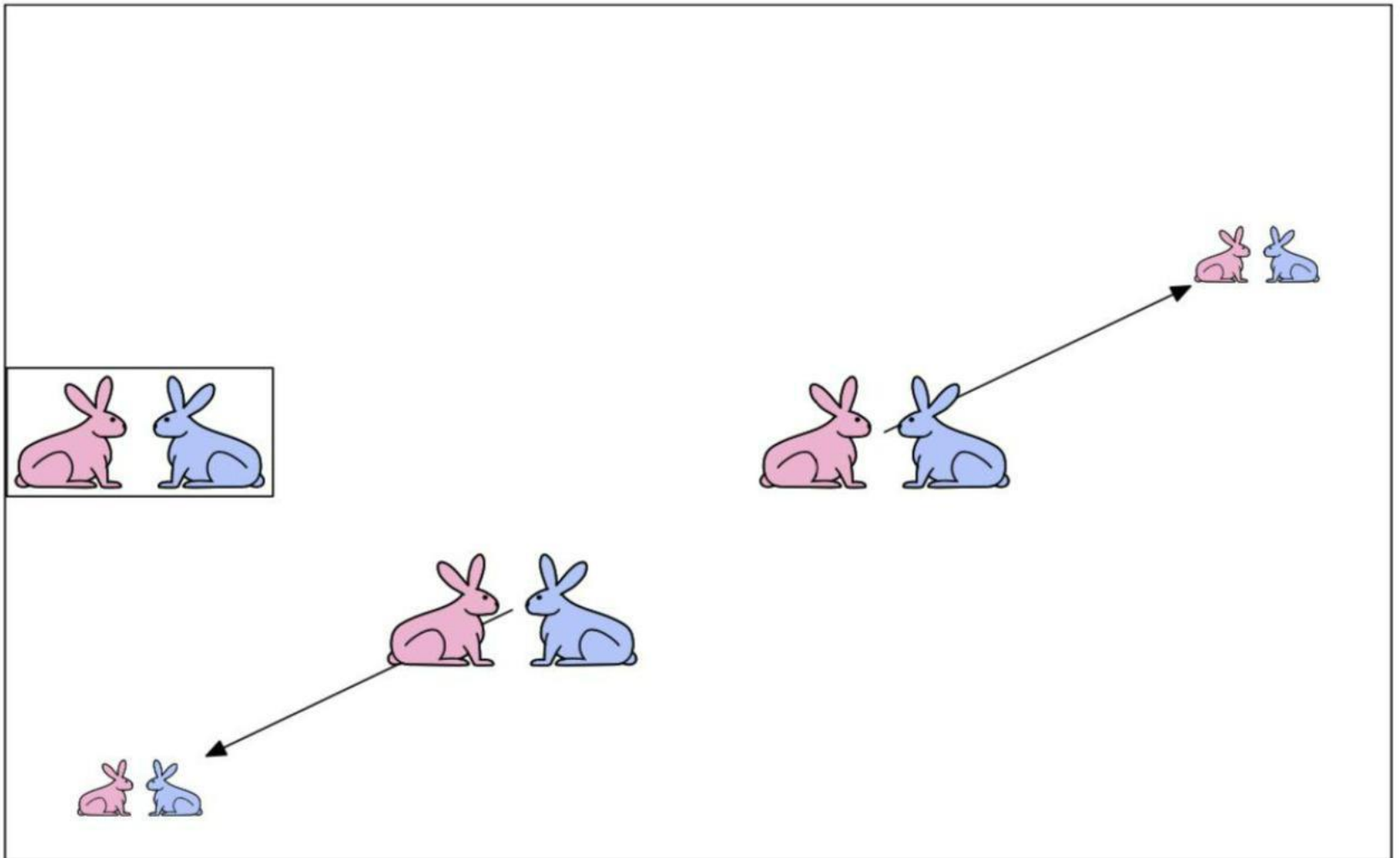
Rabbits



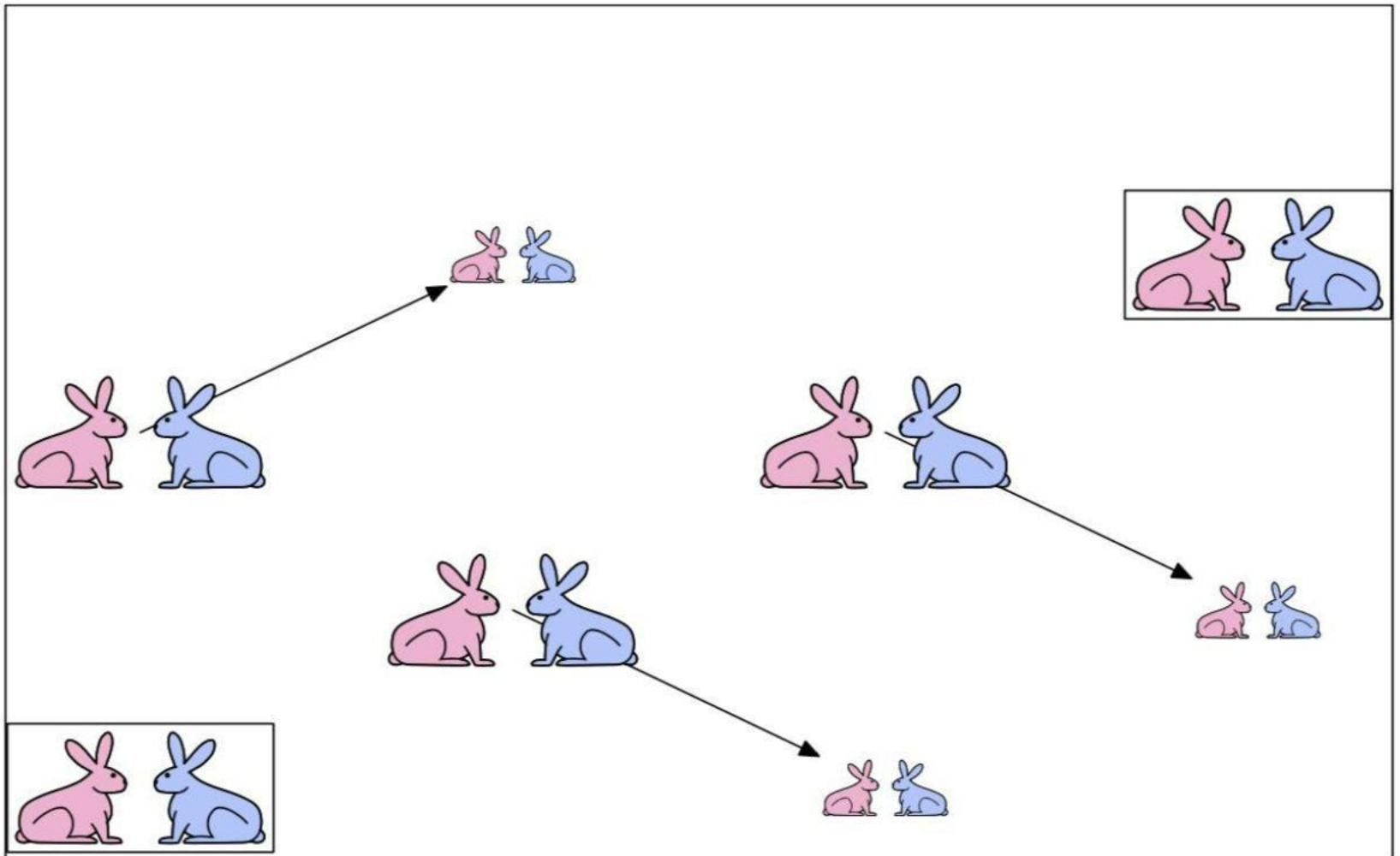
Rabbits



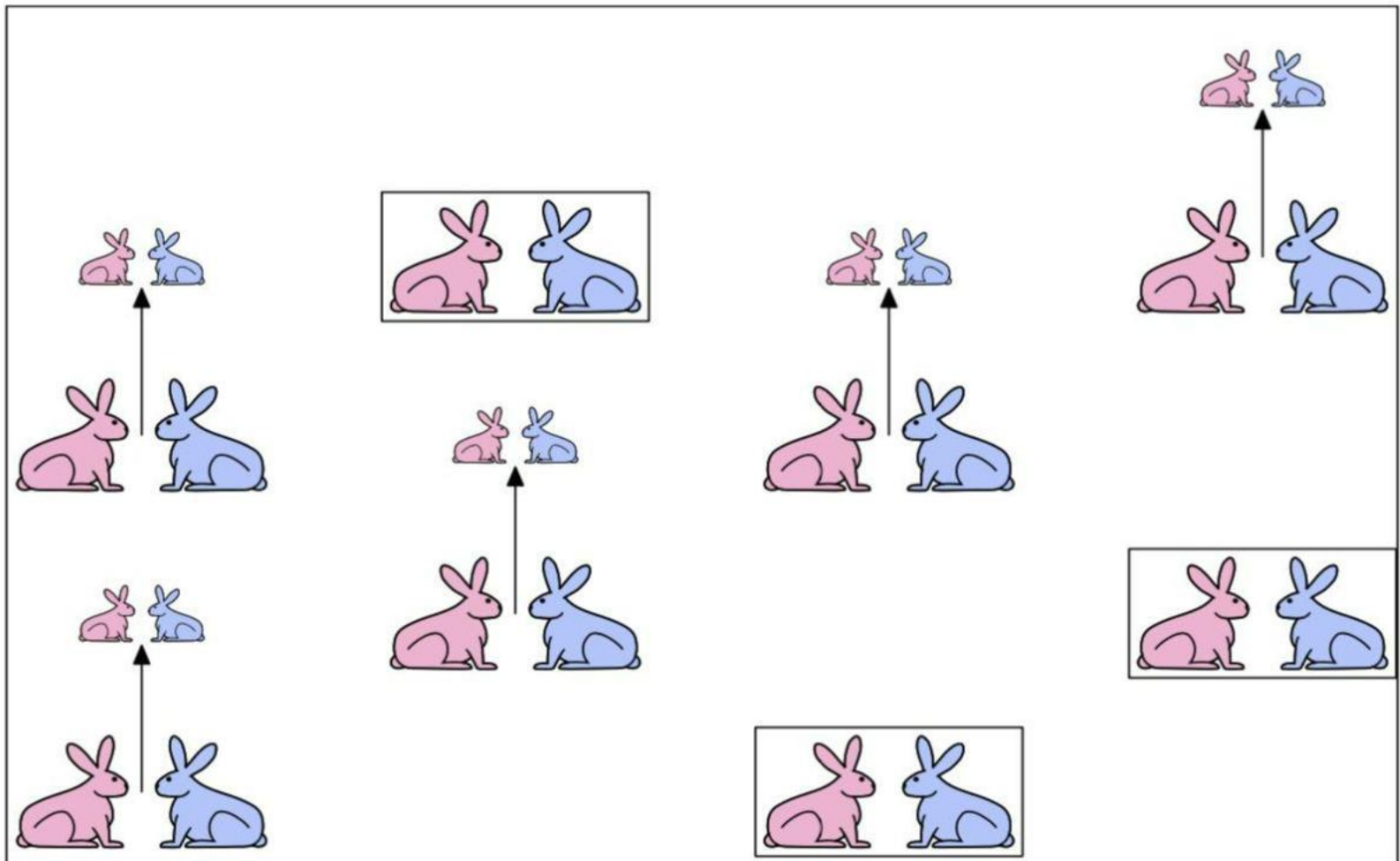
Rabbits



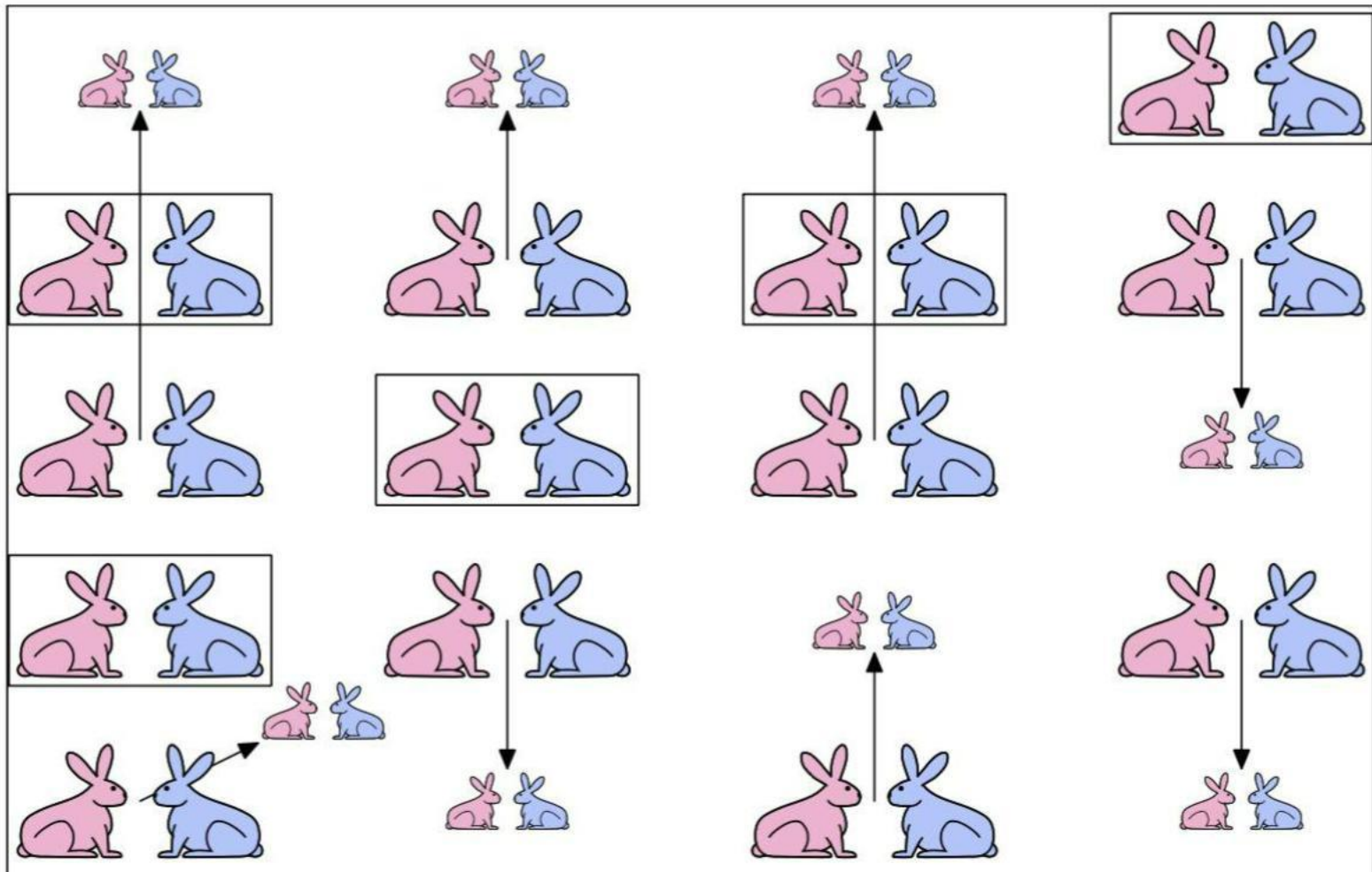
Rabbits



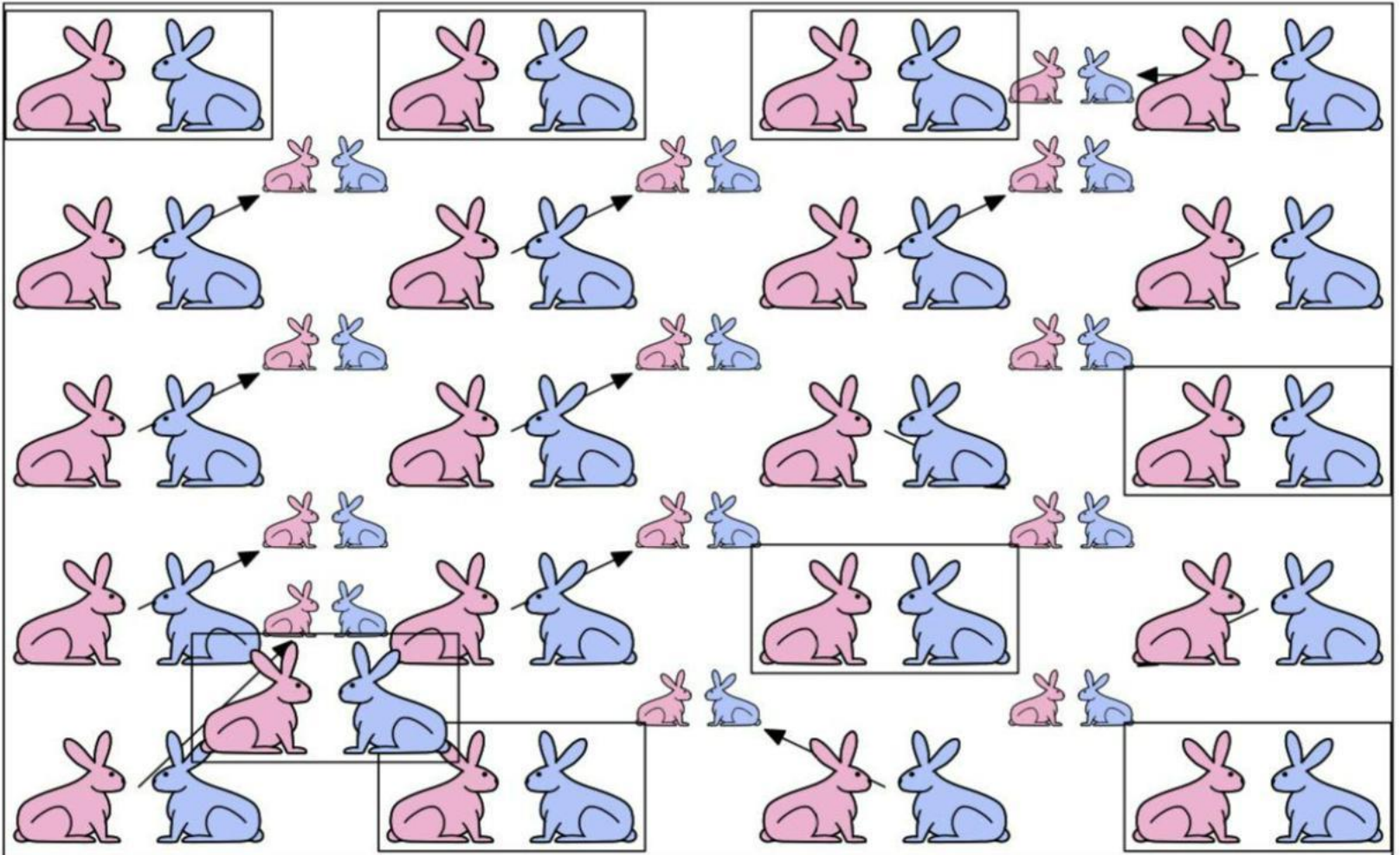
Rabbits



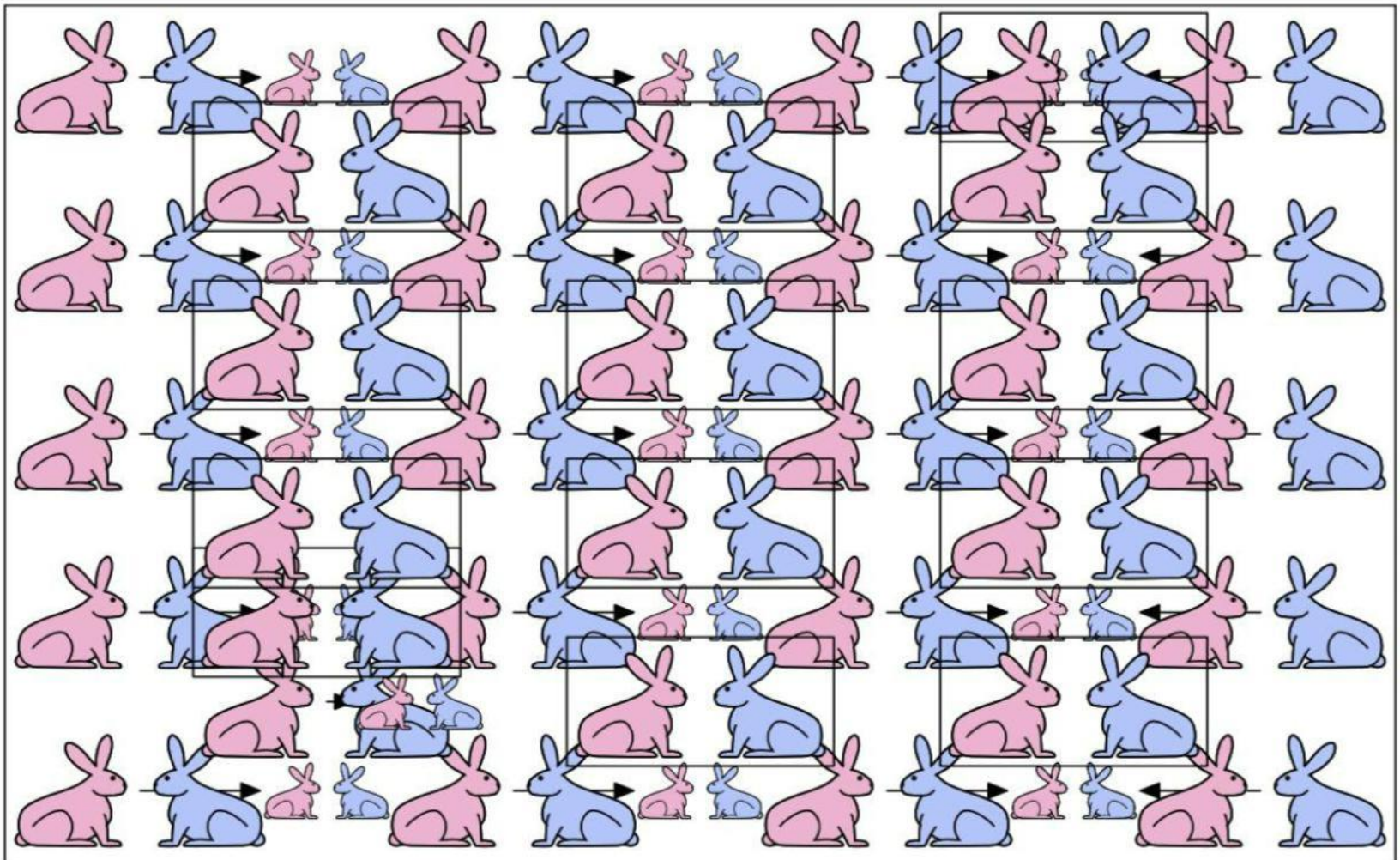
Rabbits



Rabbits



Rabbits



- [illegible]

[illegible]

FIBONACCI



- Үндсэн тохиолдол:
 - $\text{females}(0) = 1$
 - $\text{females}(1) = 1$
- Рекурсив тохиолдол:
 - $\text{females}(n) = \text{females}(n-1) + \text{females}(n-2)$

FIBONACCI



```
def fib(x):  
    """assumes x an int >= 0  
        returns Fibonacci of x"""  
    if x == 0 or x == 1:  
        return 1  
    else:  
        return fib(x-1) + fib(x-2)
```


IS PALINDROME?



```
def isPalindrome(s):  
  
    def toChars(s):  
        s = s.lower()  
        ans = ''  
        for c in s:  
            if c in 'abcdefghijklmnopqrstuvwxyz':  
                ans = ans + c  
        return ans  
  
    def isPal(s):  
        if len(s) <= 1:  
            return True  
        else:  
            return s[0] == s[-1] and isPal(s[1:-1])  
  
    return isPal(toChars(s))
```



DICTIONARIES

HOW TO STORE STUDENT INFO




- Мэдээлэл бүрийг тусдаа жагсаалтанд хадгалах

```
names = ['Ana', 'John', 'Denise', 'Katy']  
grade = ['B', 'A+', 'A', 'A']  
course = [2.00, 6.0001, 20.002, 9.01]
```

- Жагсаалт бүр **ижил урт**тай байна
- Жагсаалт бүрд **ижил индекс**ээр хандах бөгөөд индекс бүр ялгаатай хүнийг илэрхийлнэ

HOW TO UPDATE/RETRIEVE STUDENT INFO



```
def get_grade(student, name_list, grade_list, course_list):  
    i = name_list.index(student)  
    grade = grade_list[i]  
    course = course_list[i]  
    return (course, grade)
```

- Хэрэв олон мэдээлэл байвал эмх замбараагүй
- Олон жагсаалтыг бүгдийн аргументээр дамжуулна
- Бүх индекс нь бүхэл тоо байна
- Олон жагсаалтад засвар хийн гэдгийг санах хэрэгтэй

A BETTER AND CLEANER WAY - A DICTIONARY



- Сонирхсон зүйлээрээ индексжүүлэн (**Зөвхөн бүхэл биш**)
- Тусдаа жагсаалтууд биш **нэг өгөгдлийн бүтэц** ашиглана

A list

0	Elem 1
1	Elem 2
2	Elem 3
3	Elem 4
...	...

index

element

A dictionary

Key 1	Val 1
Key 2	Val 2
Key 3	Val 3
Key 4	Val 4
...	...

custom
index by
label

element

A PYTHON DICTIONARY



- Хос өгөгдөл хадгална
 - Түлхүүр
 - утга

'Ana'	'B'
'Denise'	'A'
'John'	'A+'
'Katy'	'A'

custom
index by
label

element

my_dict = { } *empty
dictionary*

grades = { 'Ana': 'B', 'John': 'A+', 'Denise': 'A', 'Katy': 'A' }

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
key1 val1 key2 val2 key3 val3 key4 val4

DICTIONARY LOOKUP



- Жагсаалтыг индекслэхтэй ижил
- Түлхүүрийг хайна
- Түлхүүрт харгалзах утгыг буцаана
- Хэрэв түлхүүр олдохгүй бол алдаа өгнө

'Ana'	'B'
'Denise'	'A'
'John'	'A+'
'Katy'	'A'

```
grades = {'Ana': 'B', 'John': 'A+', 'Denise': 'A', 'Katy': 'A'}
```

```
grades['John']      → evaluates to 'A+'
```

```
grades['Sylvan']    → gives a KeyError
```

DICTIONARY OPERATION



'Ana'	'B'
'Denise'	'A'
'John'	'A+'
'Katy'	'A'
'Sylvan'	'A'

```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
```

- Элемент **НЭМЭХ**

```
grades['Sylvan'] = 'A'
```

- Dictionary-д түлхүүр байгааг **шалгах**

```
'John' in grades      → returns True  
'Daniel' in grades   → returns False
```

- Элемент **устгах**

```
del (grades['Ana'])
```


DICTIONARY OPERATION



'Ana'	'B'
'Denise'	'A'
'John'	'A+'
'Katy'	'A'
'Sylvan'	'A'

```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
```

- Бүх түлхүүрийг авах

```
grades.keys() → returns ['Denise', 'Katy', 'John', 'Ana']
```

- Бүх утгыг авах

```
grades.values() → returns ['A', 'A', 'A+', 'B']
```

DICTIONARY KEYS and VALUES



- Утга
 - Дурын төрөл (**immutable and mutable**)
 - **Давхардаж болно**
 - Утга нь жагсаалт болон өөр Dictionary байж болно
- Түлхүүр
 - **Давхардахгүй**
 - Immutable type (int, float, string, tuple, bool)
 - Float төрлийг түлхүүрээр ашиглахад анхаарах
 - Түлхүүр эсвэл **утгаар нь эрэмблэхгүй**

```
d = {4:{1:0}, (1,3):"twelve", 'const':[3.14,2.7,8.44]}
```

LIST


vs

DICTIONARY



- Эрэмдлэгдсэн элементүүдийн жагсаалт
 - Элементийг индексээр нь хайна
 - Индексүүд нь эрэмбэтэй
 - Индекс нь бүхэл
- Түлхүүрийг утга руу харгалзуулна
 - Нэг зүйлийг нөгөөгөөр нь хайх
 - Түлхүүр нь эрэмбэлэгдээгүй байна
 - Түлхүүр нь өөрчлөгдөхгүй төрөл байна


CREATING DICTIONARY



```
def lyrics_to_frequencies(lyrics):  
    myDict = {}  
    for word in lyrics:  
        if word in myDict:  
            myDict[word] += 1  
        else:  
            myDict[word] = 1  
    return myDict
```

can iterate over list
can iterate over keys
in dictionary
update value
associated with key


USING THE DICTIONARY



```
def most_common_words(freqs):  
    values = freqs.values()  
    best = max(values)  
    words = []  
    for k in freqs:  
        if freqs[k] == best:  
            words.append(k)  
    return (words, best)
```

this is an iterable, so can
apply built-in function
can iterate over keys
in dictionary

LEVERAGING DICTIONARY PROPERTIES



```
def words_often(freqs, minTimes):
    result = []
    done = False
    while not done:
        temp = most_common_words(freqs)
        if temp[1] >= minTimes:
            result.append(temp)
            for w in temp[0]:
                del(freqs[w])
        else:
            done = True
    return result
```

can directly mutate
dictionary; makes it
easier to iterate

```
print(words_often(beatles, 5))
```