DS5010

Intro to programming for Data Science

LECTURE 7

TODAY

- review session 1
- review session 2
- review session 3
- review session 4
- review session 5
- review session 6

REVIEW SESSION 1

- basic python objects and operations
- conditionals
- branching

Basic data types in python

Int Any integer.

Float Floating point number (64 bit precision)

Complex Numbers with an optional imaginary component.

Bool True, False

Str A sequence of characters (can contain unicode characters).

Bytes A sequence of unsigned 8-bit entities, used for manipulating binary data.

NoneType (None) Python's null or nil equivalent, every instance of None is of NoneType.

Basic operators in python

= Assignment

+ Addition

- Subtraction

* Multiplication

/ Division

// Floor Division

% Modulo

** Power

Comparison Operators

== Equal To

> Greater Than

>= Greater Than or Equal To

< Less Than

<= Less Than or Equal To

!= Not Equal

Boolean Operators and or not

Expressions

Combine operators and objects

<object> <operator> <object>

Are these legal python expression?

5 + “abc”

5 >= 2 == 2

if (x < y < z) :

“a” + “b”

Input and Output

Collect data from the user, you can also provide a prompt

input("Type anything... ")

Print – can accept a comma separated list of values

print(”x is “, x_int)

… or you can concoct your own string with the + operator

print("x = " + “xxx”)

String data objects

Every string object has a length len(“abs”) and each character in the string has a position. The position can be used to extract characters from the string.

Positions are 0 based, and the last character can also be indexed with -1

You can extract a substring by using the colon index operator

string[start:end-1] end-start are the number of characters extracted

string = “abcdefg”

string[2:4] == ???

string[:4] == ???

string[4:] == ???

Boolean operators and bool casting

A	B	not A	A and B	A or B
True	True	False	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	Fakse

Other data types can be cast to the bool type. There are specific values that map to the False values

Type	False value
int	0
float	0
complex	0+0i
str	“”
list	[]
dict	{}
tuple	()
set	set()
NoneType	None

Branching in a program

```
if <condition>:
    <expression>
    <expression>
    ...
```

One condition tested

```
if <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

One condition tested with a branch on the condition

```
if <condition>:
    <expression>
    <expression>
    ...
elif <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

Multiple conditions tested but only one set of instructions executed

What output is produced?

a = 10

b = 3

c = 5

Order matters!!

```
If ( a < c):
    print(“First condition”)
elif (c > 5):
    print(“Second condition”)
elif (b < a):
    print(“Third condition”)
else
    print(“Default condition”)
```

REVIEW SESSION 2

ITERATION WITH **FOR** AND **WHILE** LOOPS

LIST TUPLES, DICTIONARY

MUTABLE AND IMMUTABLE OBJECTS

FILE I/O

Format for a while loop

```
while condition:
    body of loop
    body of loop
outside loop
```

```
for iterator in sequence:
    body of loop
    body of loop
outside loop
```

```
n = 0
while n < 5:
    print("Hello world")
    n = n + 1
```

```
for n in range(5):
    print("Hello world")
```

The break command terminates a loop

It allows the current loop instantiation to terminate

It only terminates one level of a loop – so will not terminate all levels of a nested loop

If j == 3 what is the next printout?

```
for i in range 5:
    print("At begin: outer loop")
    for j in range 6:
        if j == 3:
            break;
        print("pass break")
print("At end:  outer loop")
```

Tuples

Tuples are a collection of comma separated values. They are immutable – this means once assigned a value, the tuple cannot be changed. Once created, a tuple is analogous to ROM (read only memory).

An empty tuple t0 = ()

A tuple with one element is t1 = (1,)

A tuple with 5 elements t4=(“iris”, “pansy”, “sage”, “thyme”)

A nested tuple t2 = (t1,t4)

Operators on tuples

Repetition can be applied to a tuple *

Addition can be applied to a tuple +

Slicing can be applied to a tuple (n:m)

t4=(“iris”, “pansy”, “sage”, “thyme”)

What is the result of ?

t4*3

t4 + (“marigold”)

t4(2:3)

Discussion

Tuples

- Break up into groups and discuss the utility of the tuple object.
- Consider the benefit of a mutable object.

Response: Tuples

Swapping data values

Returning multiple values from a function

Ensuring a data value is not updated

The list data object

An ordered collection of elements. Similar to a string, an element can be retrieved with an index.

A list is denoted with square brackets:

Empty = [] L = [1,2,3,4]

A list is mutable, both in the elements and in its size.

A common operation is to iterate over a list.

```
val = 0

for e  in L:

    val += e
```

Some useful list methods

Method	Description
Lisr.append	Add a member to the end of the list
List.insert(index,value)	Inserts a value at a particular index
List.extend(List_too)	All values in List_too are added to List
List.remove(value)	Removes the first instance of value from List
List.pop()	Returns and removes the last element in the list
List.pop(index)	Returns and removes the element at position index
List.clear()	Remove all values from the list
List.index(value)	Returns the position of the first occurring value of value
List.copy()	Clones a List

List operation examples

L=[1,2,3,4]

L.append(5) L becomes [1,2,3,4,5]

L.remove(2) L becomes [1,2,4,5]

L.pop() L becomes [1,3,4]

T = L[:] clone a list

Delete by position, del(L[1]), removes 3 from L, L=[1,4]

The string's join method makes a string out of a list of characters
“”.join(“a”,”b”)

The list operator converts a string to a list

The split operator will split a string into a list

Dictionary data object

Associate a data value with a key value. Allows you to index a data collection via a key value.

An empty dictionary is represented as {} If tel =

Key	Value
Robert	555-5555
Lisa	555-1010
Xi	555-1212

Then tel{‘Kay’} will generate a lookup error.

Operations on a dictionary

Add an object

tel{“Kay”} = “555-1213”

Look up a value

‘Kay’ in tel

tel.get(“Kay”) tel.get(”Sam”)
returns the value None
tel[“Sam”] generates an error

Delete a value Del tel{“Lisa”}

tel.keys returns an iterable list of the key values

tel.values returns an iterable list of the values values

Requirement: key values must be unique and immutable

REVIEW SESSION 3

- permutations
- combinations
- user-defined functions

Permutation

Permutation of a set of objects is an ordered arrangement of those objects

Example:

The wedding party:

Bride, **G**room, **BM**other, **BF**ather, **GM**other, **GF**ather, best**M**an, maid**H**onor, **U**sher**1**, **U**sher**2**, bride's**M**aid**1,** bride's**M**aid**2**

How many ways are there of arranging them in a row for a picture?

12!

Permutation with selection

What if we wanted to select a subset of elements from a set – where order still matters . The is typically called an **r-Permutation,** an ordered arrangement of r elements of a set.

I want to take a photo of 6 people in the wedding party. How many different permutations are there?

P(n,r) = n(n-1)(n-2) . . . (n – r + 1) = n! / (n-r!)

In this example duplicates are not allowed. What if duplicates are allowed?

Permutations with repetitions

Let’s say there are 24 flavors of ice cream as Rancatore’s. The order that I taste the ice cream is important since it may influence my next choice of ice cream tasting. How many permutations are there if I am allowed to taste 6 flavors and I am allowed to sample a flavor more than once?

24**6, Since I have 6 samples and for each sample I have 24 choices.
First choice 24 to choose from , second choice 24 to choose from , … sixth choice 24 24*24*24*24*24*24

Combination

Combination of a set of objects is an arrangement of those objects where **order does not matter**

r-combination of elements of a set is an unordered selection of r elements of the set, i.e. a subset of the set with r elements.

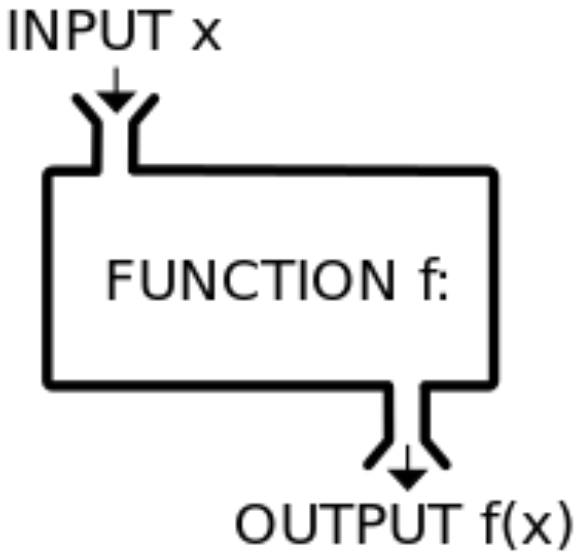
Example:

Eight members of the wedding party are to do a traditional circle dance. How many different groups of eight can be selected?

12! / 4!*8!

$$\text{If } 0 \le r \le n \text{ are integers, } C(n,r) = \binom{n}{r} = \frac{n!}{r!(n-r)!}.$$

Functions in python



Functions introduce the concepts of scope, abstraction, decomposition, and modularity

Break up into groups to discuss these concepts in respect to functions

Response Function

Functions limit the scope for the variables within it. Variables inside the function are not known outside of the function **(Scope)**

A function can be used by a programmer without the programmer knowing how it works. The programmer just needs to know the function's input values (parameters) and its output values (**Abstraction)**

A function allows the code in the function to be used (or called) from different points in the program **(Modularity)**

It allows a large problem to be broken up into simpler problems **(Decomposition)**

Example

Use the def keyword to define a function

```
def action(ready):
    if (ready) :
        print('Time to go')
        return True
    return False


action(True)
```

Should return a value, since the default is to return the None value

Arguments as functions

Functions can be written such that accepts a function argument. This allows a programmer to actually change the functionality of the function with its argument.

Python provides built in functions that accept functions as arguments. The filter() and the map() function

REVIEW SESSION 4

- recursion
- induction
- lambda expressions

Inductive proof

Start with a base case – show what you are trying to prove is true for the base case, usually n = 1 (but can be a different value).

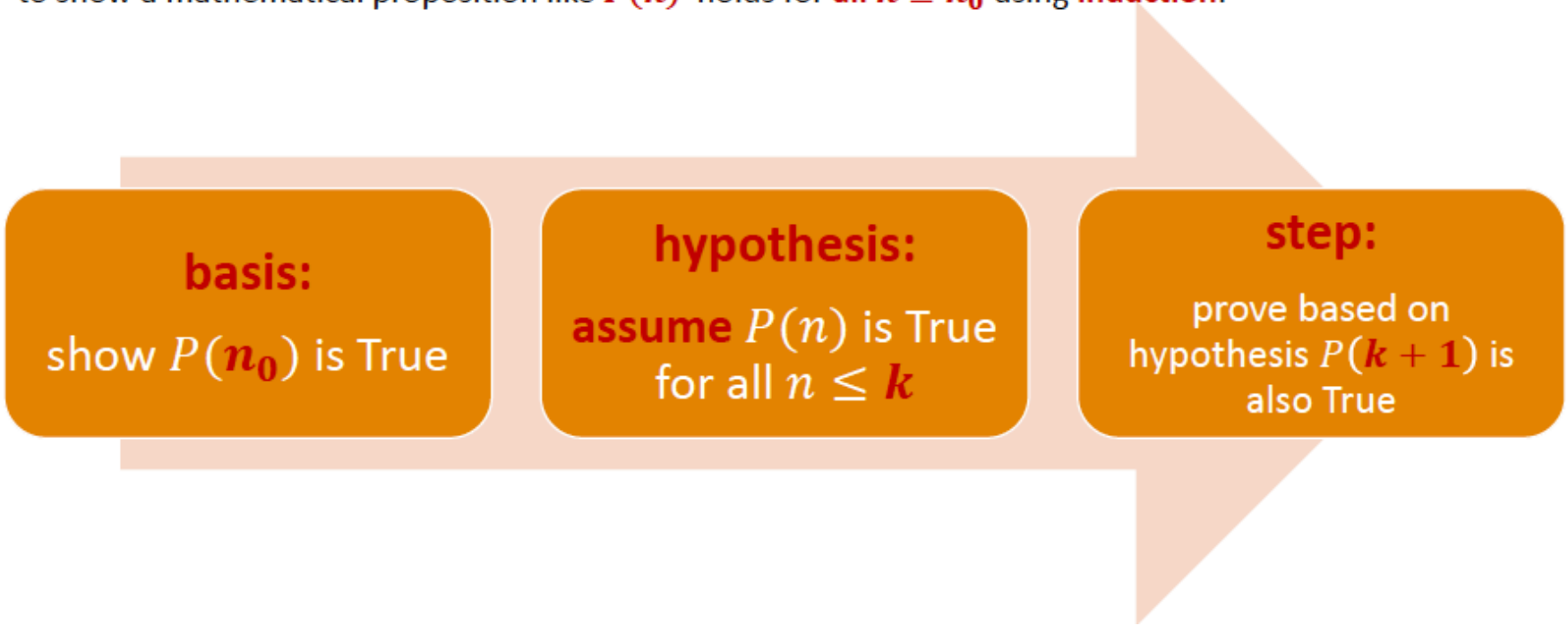
Show that if **n=k** is true, then **n=k+1** is also true

We typically assume that n=k is true, we use this assumption to show that it is true for n=k+1

Like the domino affect – one falls they all fall

Steps in induction

to show a mathematical proposition like $P(n)$ holds for **all** $n \geq n_0$ using **induction**:



Example proof

Proof by induction: is 3**(n) -1 a multiple of 2 ?

Base case: let n = 1, 3**1 = 1 so 3- 1 = 2 , 2 is a multiple of 2.

Assume: 3**(k) -1 is a multiple of 2 – show that 3**(k+1) -1 is a multiple of 2.

3**(k+1) +1 == 3*3**(k) -1 , we can break 3*3**(k) into 1*3**(k) + 2*3**(k) . The first term 3**(k) -1 is a multiple of 2 due to the assumption. The second term is a multiple of 2, since 2 is a factor of the term.

Hence we showed that 3**(n) -1 a multiple of 2 for all n

Recursion

Recursion is related to induction, since it allows us to define a concept in terms of itself. This is especially useful for series of numbers or for algorithms that perform a specific operation until it hits a terminating state.

Recursion:

find the solution for the base case

assume you have the solutions for all of the smaller inputs

show the bigger problem can be solved using the solution to the smaller problems

Divide and conquer

Break the problem up into smaller problems

Have a function call itself to simplify the problem (via the passed arguments) that needs to be solved

Function terminates at a base case

Each invocation of the function creates a separate scope for the different instantiations of the function variables

When a function terminate control flow is returned to the calling level

Code Review

```
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))
```

Higher Order Functions

A higher order function is a function that takes a function as an argument

There are built-in higher order functions in python:

map – apply a function to an iterable data object and return it as a result

filter – filter applies a function to each element in an iterable, if the function returns True, then that element is returned in the result, if returns False, not part of the result

Anonymous functions

Lambda expressions allows one to create a function without a name. This is useful to do if you are passing the function to another function. The definition of the function is the value that gets returned

```
def square(y):
    return y*y;

print(square(5))

g = lambda x: x*x
print(g(5))
```

A lambda is useful with map and filter

```
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]

final_list = list(filter(lambda x: (x%2 != 0) , li))

print(final_list)
```

```
map(lambda x : x*2, [1, 2, 3, 4])
```

```
map(lambda x, y : x*y,

[1, 2, 3, 4], [5,6,7,8])
```

REVIEW SESSION 5

- what are objects
- classes of objects

Objects in the real world

Humans have been classifying objects in the real world since the dawn of day. These categories or types group instances of objects by similar characteristics or attributes

Attributes can also describe behavior, what are the types of actions object instances in this class can do

The **state** of an object may **change/evolve over** time, but that object remains the same existence! This change is usually done by means of behavioral attributes.

If we are grouping objects into a type, we will want to determine how similar these objects are to each other

Objects in python

Python supports many different types of objects

Every object has:

- a type
- an internal **data representation**(primitive or composite)
- a set of procedures for **interacting** with the object

- object is an **instance** of a type
  - For example, 1234 is an instance of an int, "hello" is an instance of a string

Discussion

Instantiation

- Break up into groups and discuss the difference between object instantiation and and object design .

Response

Use of an object versus encapsulating the definition of an object

Program design incorporates both activities, one process provides feedback to the other process

Data encapsulation

Code Review

- Review the Objects:
  - Coordinate and Fraction
  - 7Example_class.py

REVIEW SESSION 6

- Defining and using user-defined classes
- Object oriented programming

HIERARCHIES

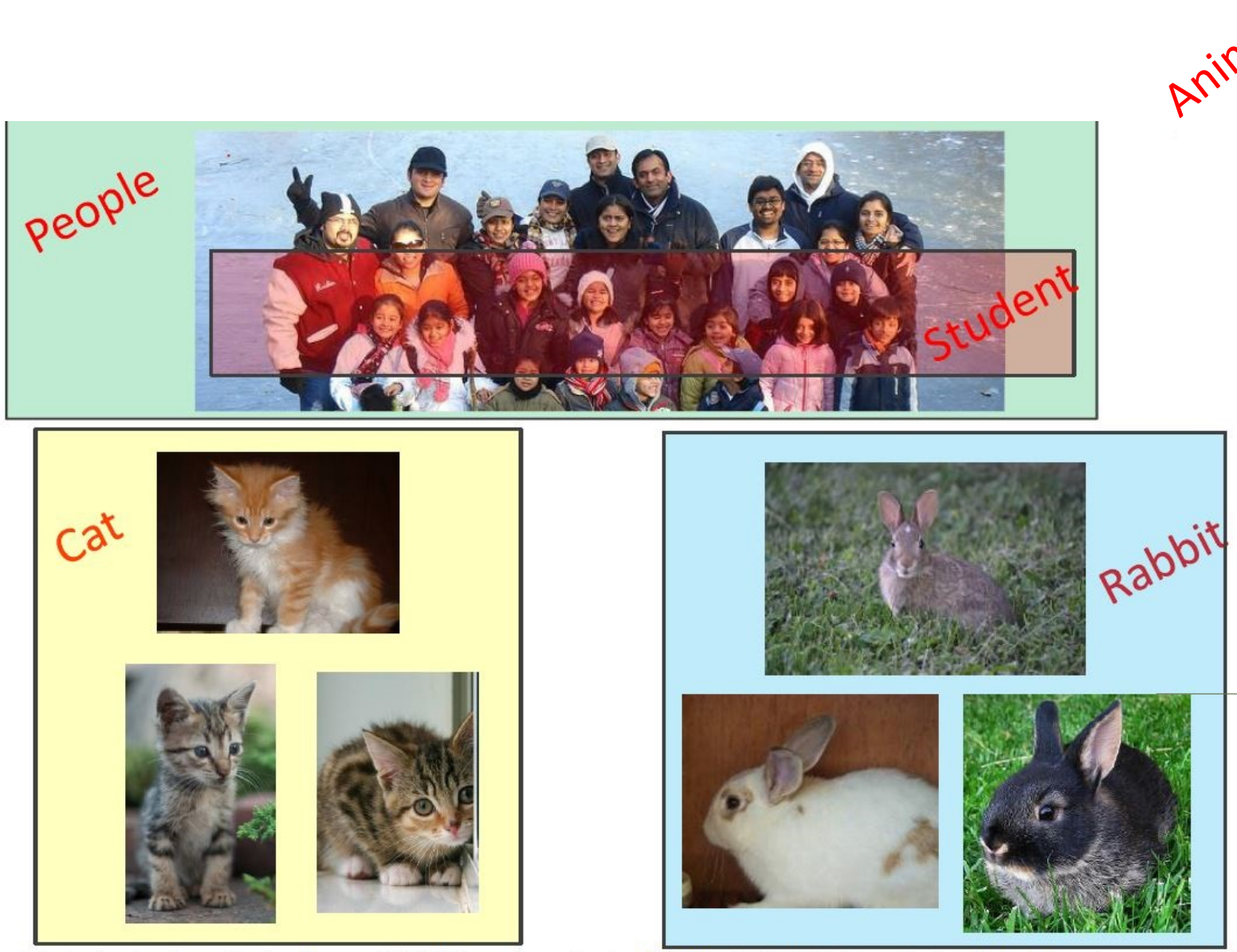
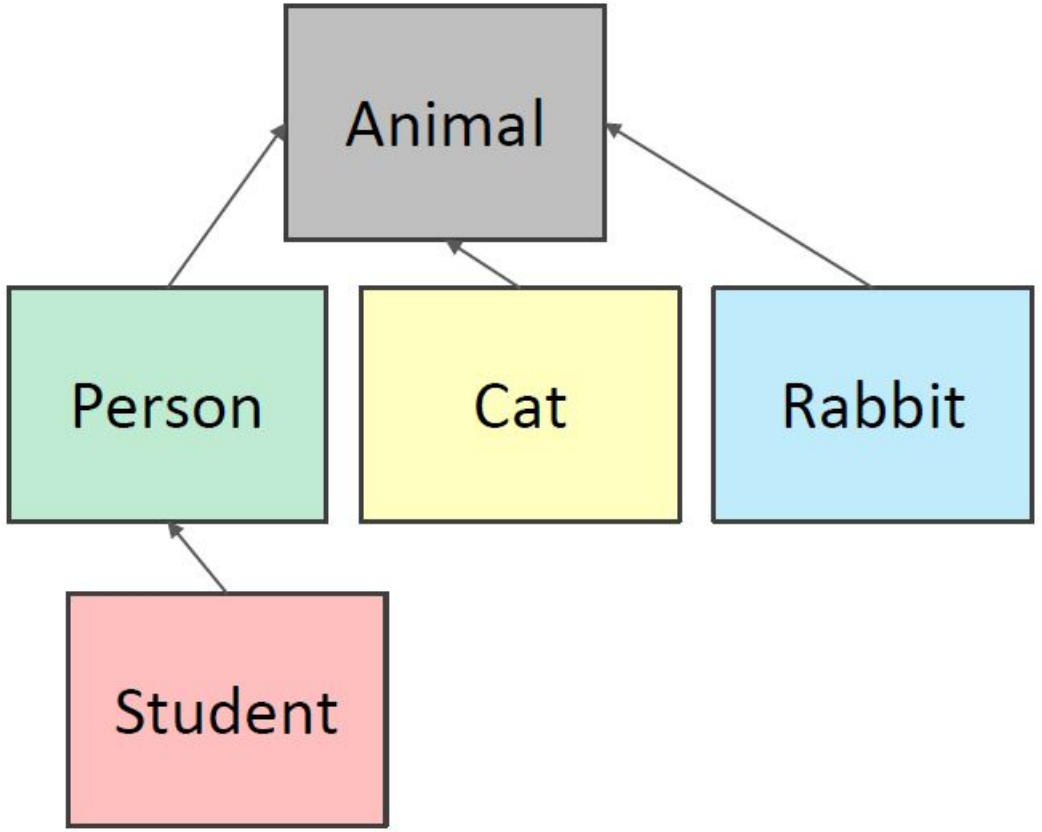


Image Credits, clockwise from top: Image Courtesy Deeeep, CC-BY-NC. Image Image Courtesy MTSOfan, CC-BY-NC-SA. Image Courtesy Carlos Solana, license CC-BY-NC-SA. Image Courtesy Rosemarie Banghart-Kovic, license CC-BY-NC-SA. Image Courtesy Paul Reynolds, license CC-BY. Image Courtesy Kenny Louie, License CC-BY. Courtesy Harald Wehner, in the public Domain.

HIERARCHIES

- **parent class**(superclass)
- **child class**(subclass)
  - **inherits** all data and behaviors of parent
  - **add** more **info**
  - **add** more **behavior**
  - **override** behavior



Discussion

Object Orient Programming

- Break up into groups and describe the benefits of object orient programming.

Response

- create your own **collections of data**
- **organize** information
- **division** of work
- access information in a **consistent** manner
- address complexity by stratifying it into **layers**
- like functions, classes are a mechanism for **decomposition** and **abstraction** in programming

Code Review

- Review the animal hierarchy:
  - 7Example_hierarchy.py

Discussion

Stack vs. Queue

- Discuss the definition of a stack and a queue. Identify their similarities and differences

Response

Stack: a collection of items where the last added item is the first retrieved item. All other items are masked by the item on the top.

Queue: a **collection** of items where the **first** added item is the **first** item that can be retrieved. All other items are masked by the item on the top.

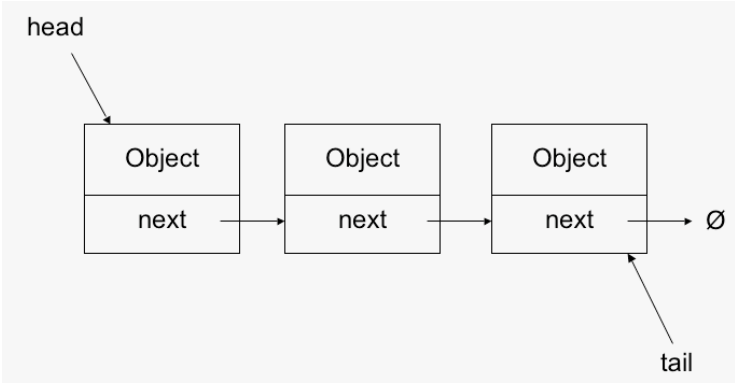
Code Review

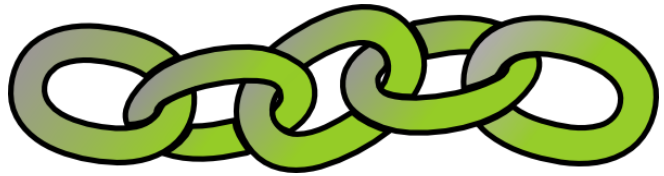
- Review the stack and queue class:
  - 7Example_stack_queue.py

Discussion

Linked list

- Discuss the definition of a linked list. Discuss how it varies from a simple built in list, a stack and a queue.





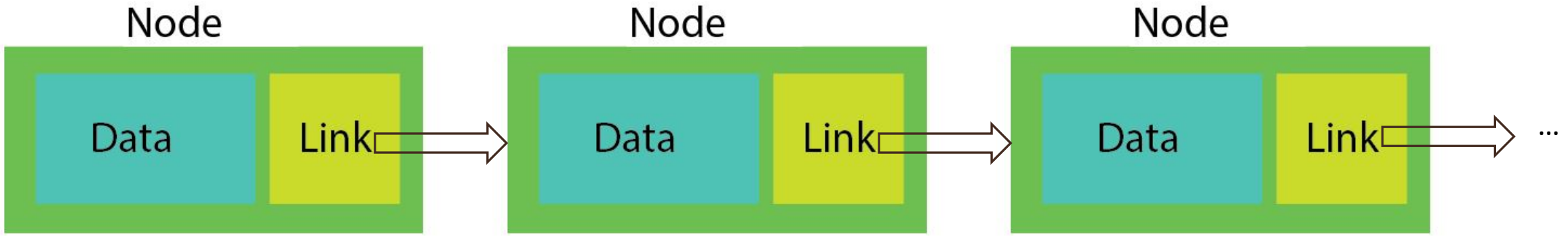
Response

Linked list: a data structure consisting of a collection of **nodes** which together represent a sequence. It is not contiguous in memory.

Benefit? the memory reserved for the link list can be increased or reduced at runtime. items can be added anywhere in the sequence

LINKED LIST is a CHAIN of NODES

- each node contains: **data**, and a **reference** (in other words, a link) to the next node in the sequence



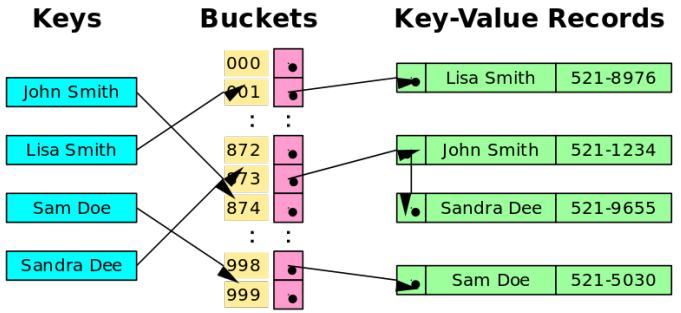
Code Review

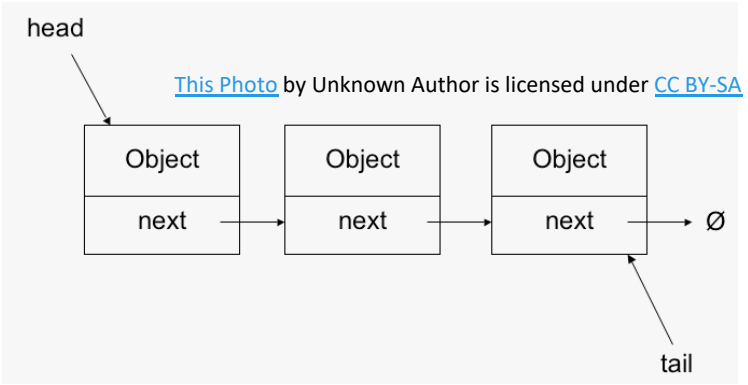
- Review the stack and queue class:
  - 7Example_stack_queue.py

Discussion

Hash table

- Discuss the utility of a hash table. Discuss how it varies from a linked list, simple built in list, a stack and a queue.



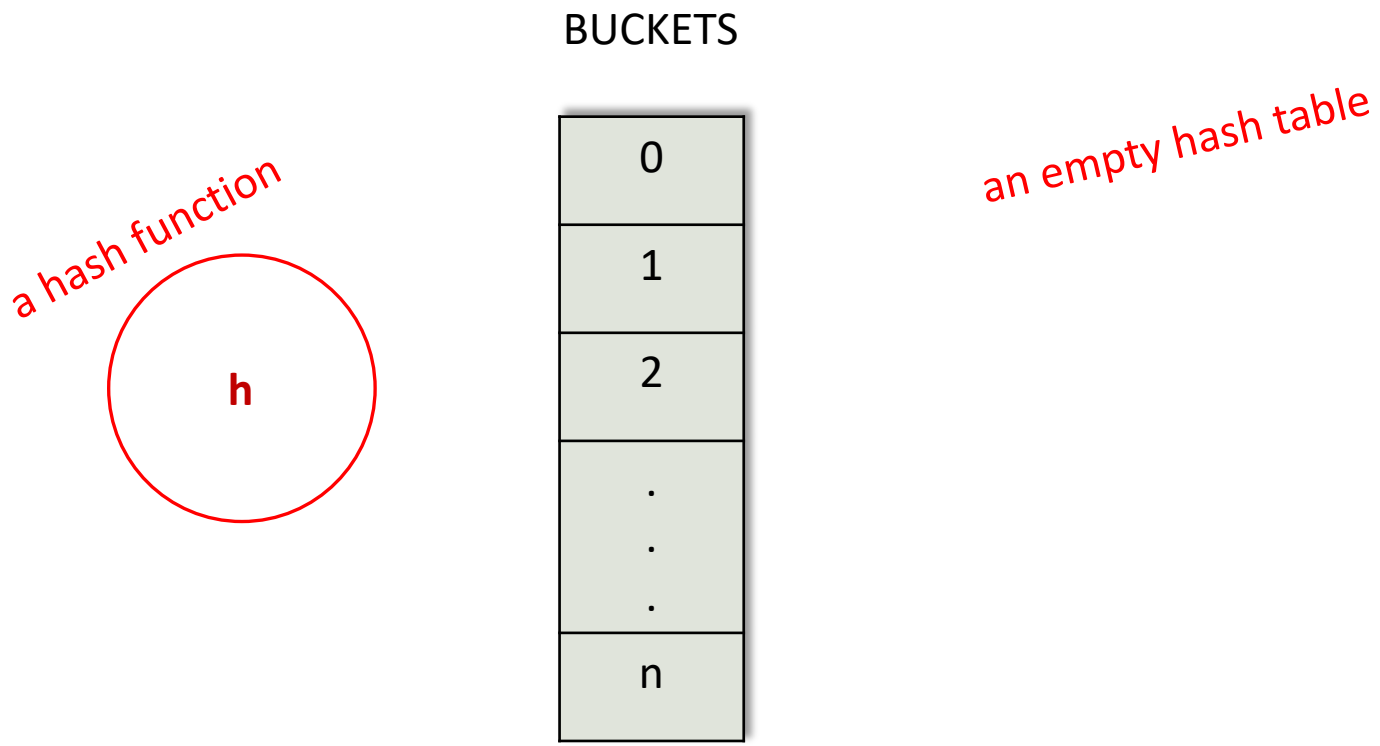


Response

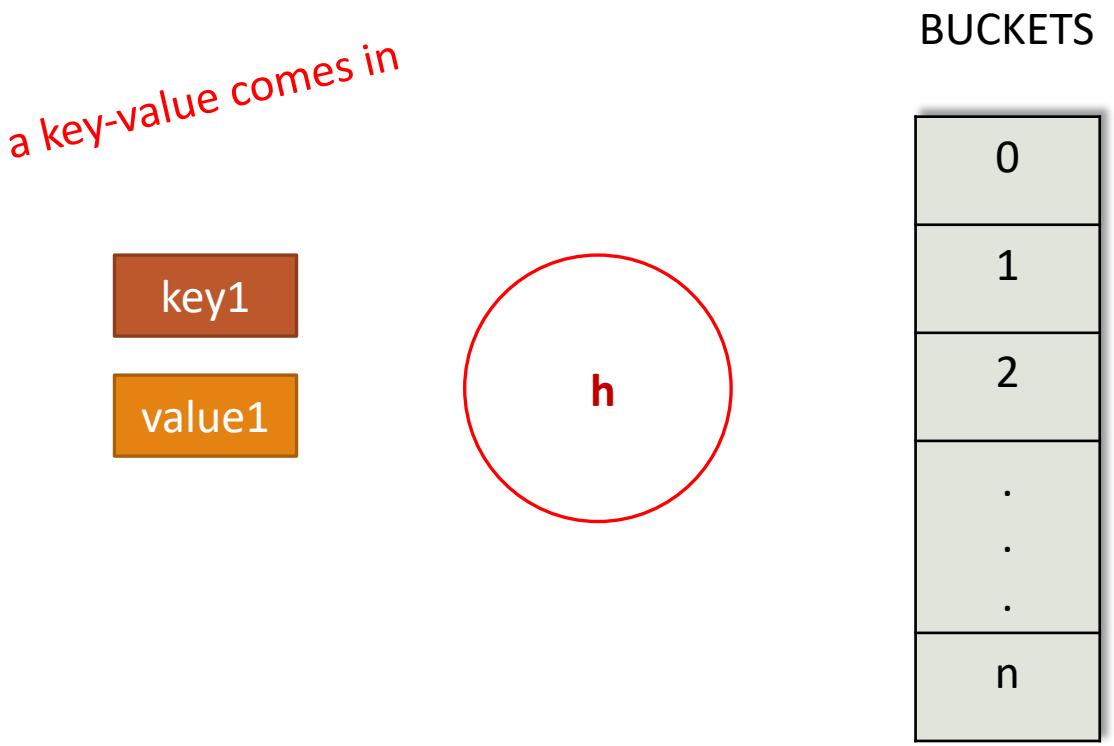
- Hash table: a structure that can **map keys to values**. A hash table uses a **hash function** to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found. It uses a hash function to disperse the entries across the buckets. Like a linked list, items are not stored in sequence fashion but the hash buckets can be contiguous.. By using a good hash function, hashing can work well. Under reasonable assumptions, the average time required to search for an element in a hash table is **O(1)**.

Quick Lookup !!

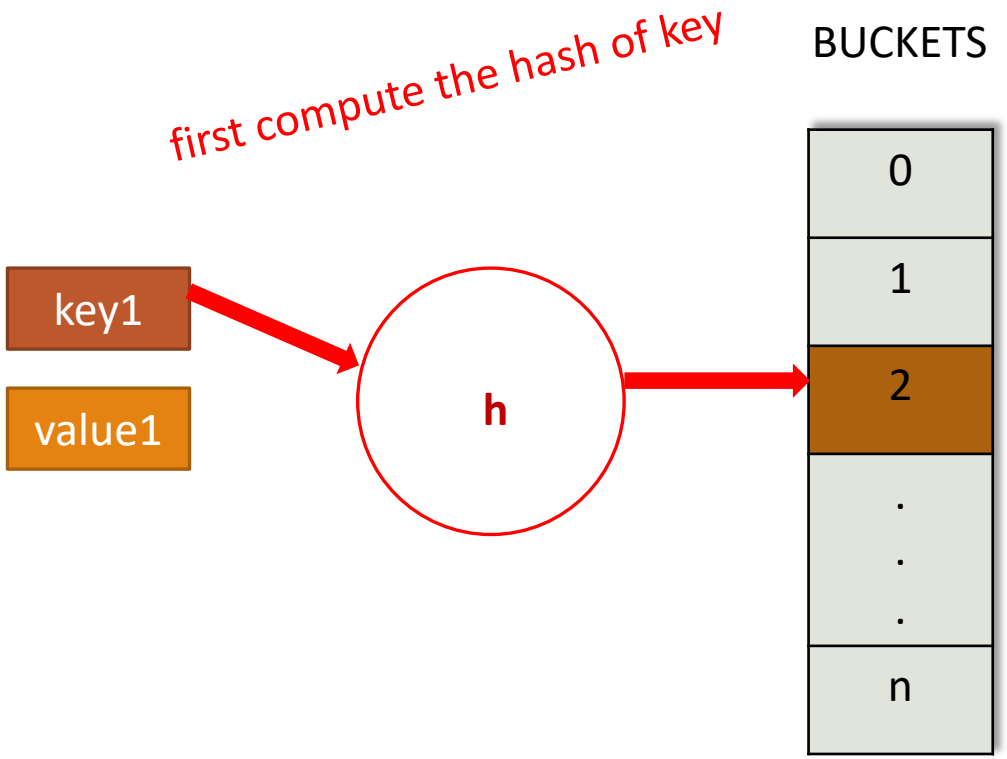
INTERNAL STRUCTURE



INTERNAL STRUCTURE

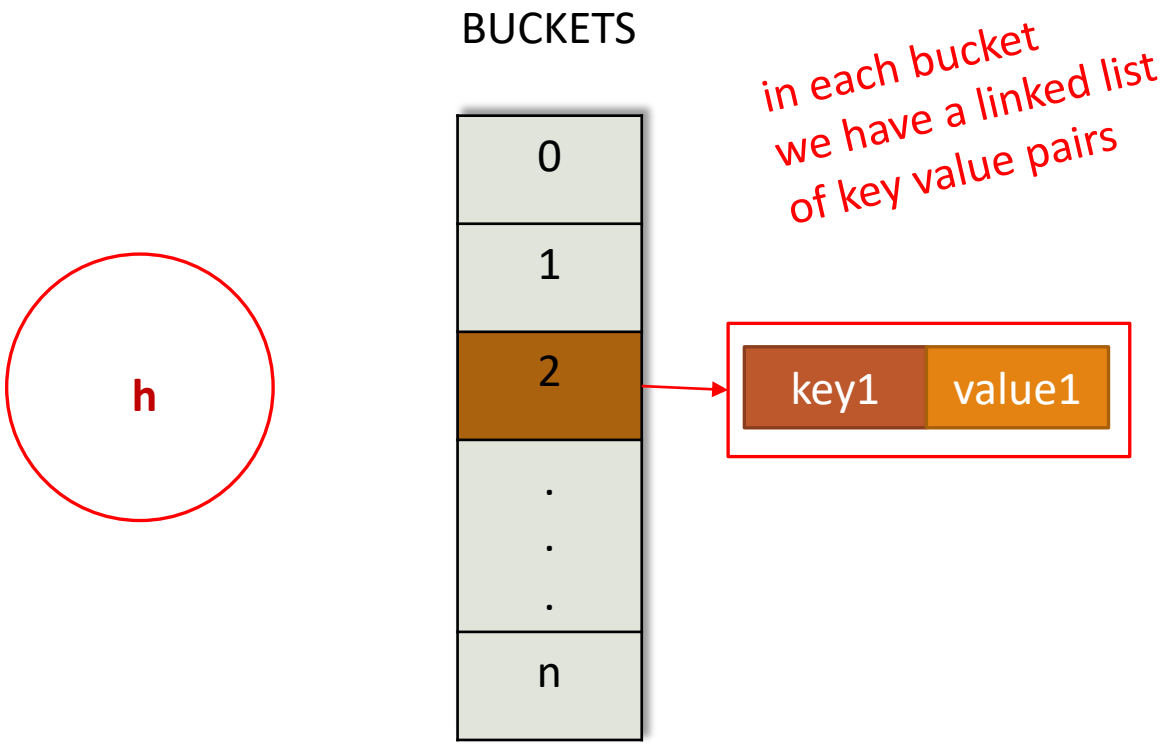


INTERNAL STRUCTURE

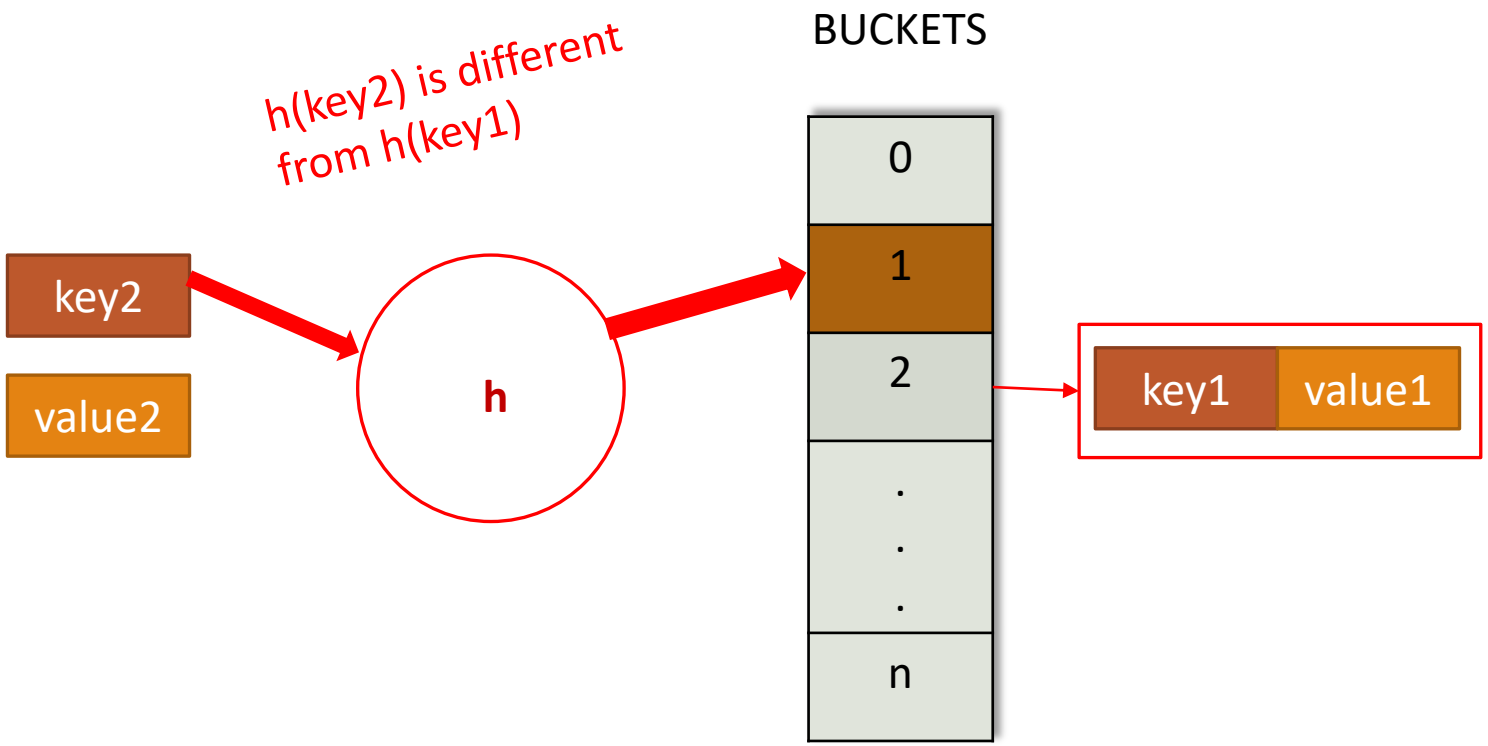


- **hash function** accepts keys as input, and outputs an **integer** which represents a bucket **index**
- the key-value pair like `(k,v)` will be put into a bucket, with `index = h(k)`

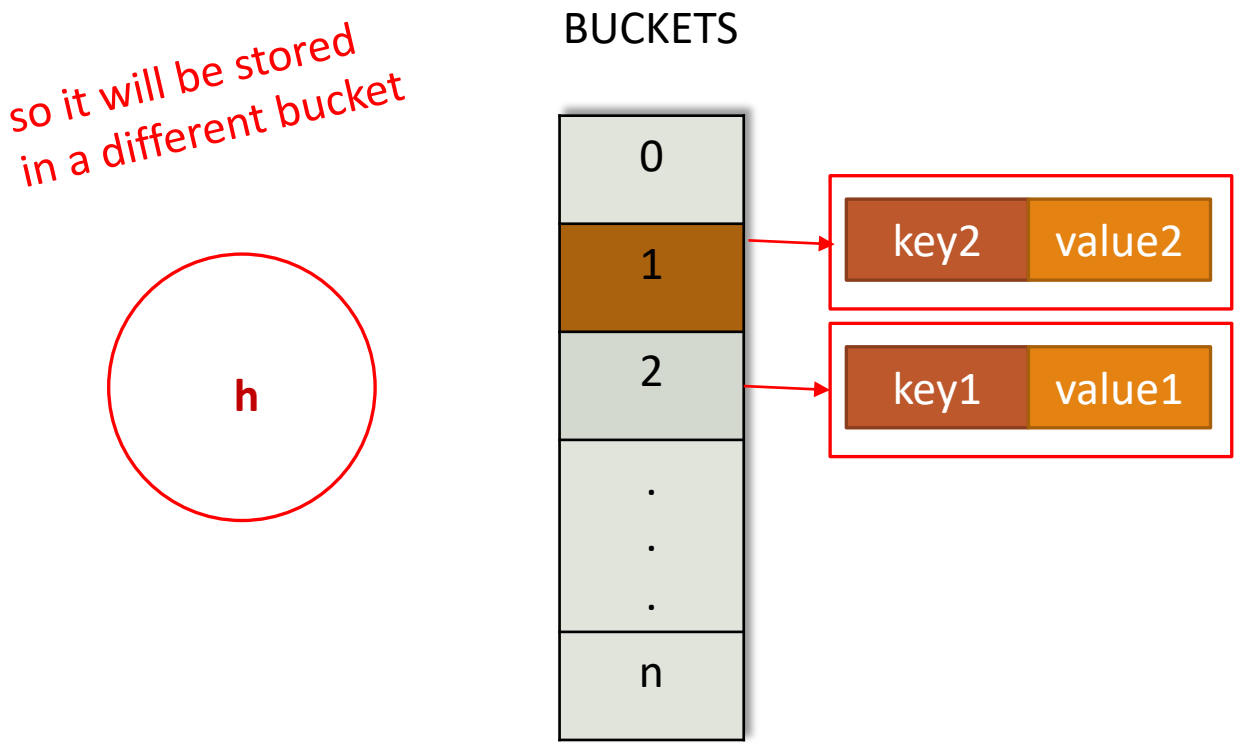
INTERNAL STRUCTURE



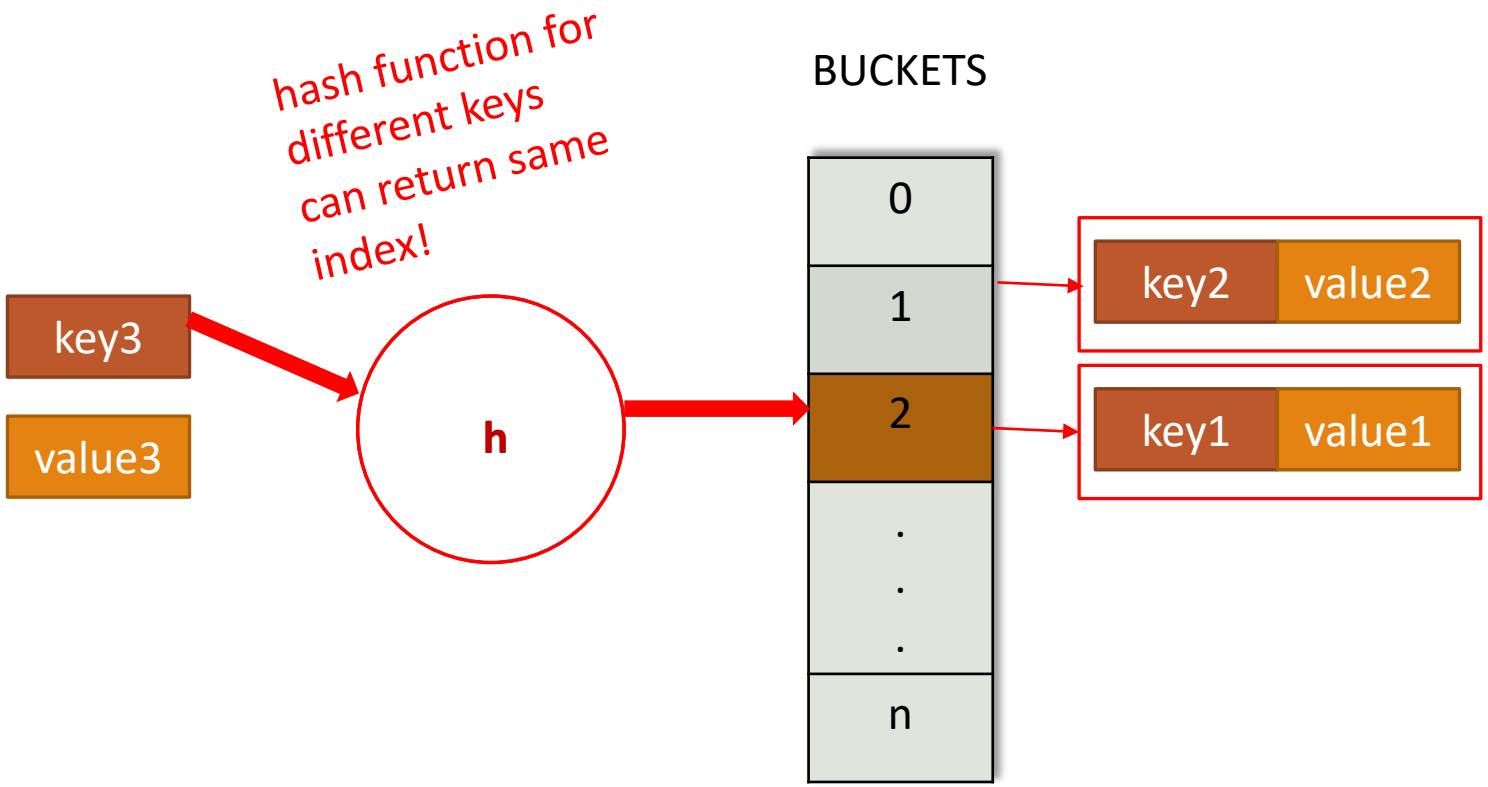
INTERNAL STRUCTURE



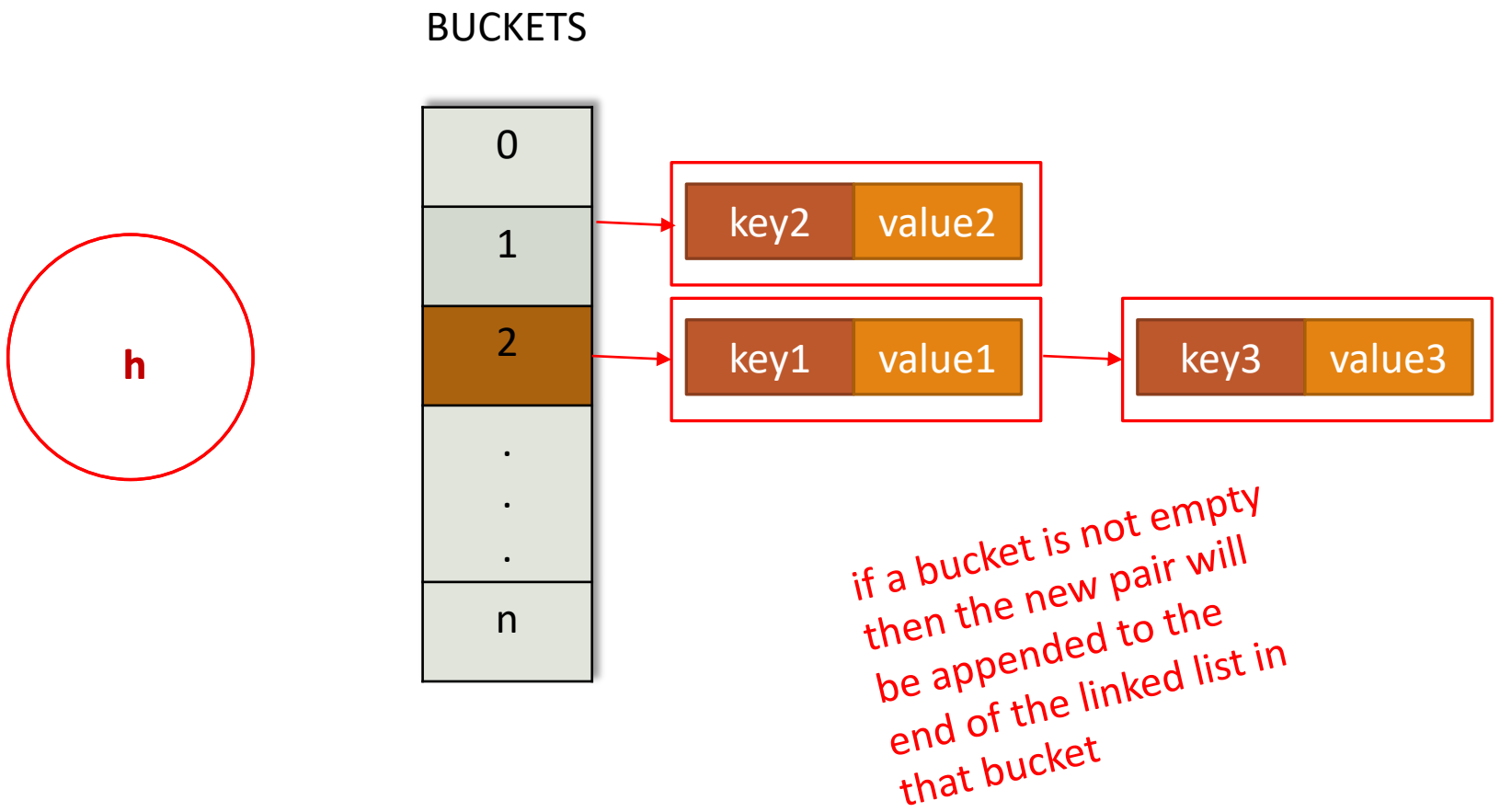
INTERNAL STRUCTURE



INTERNAL STRUCTURE

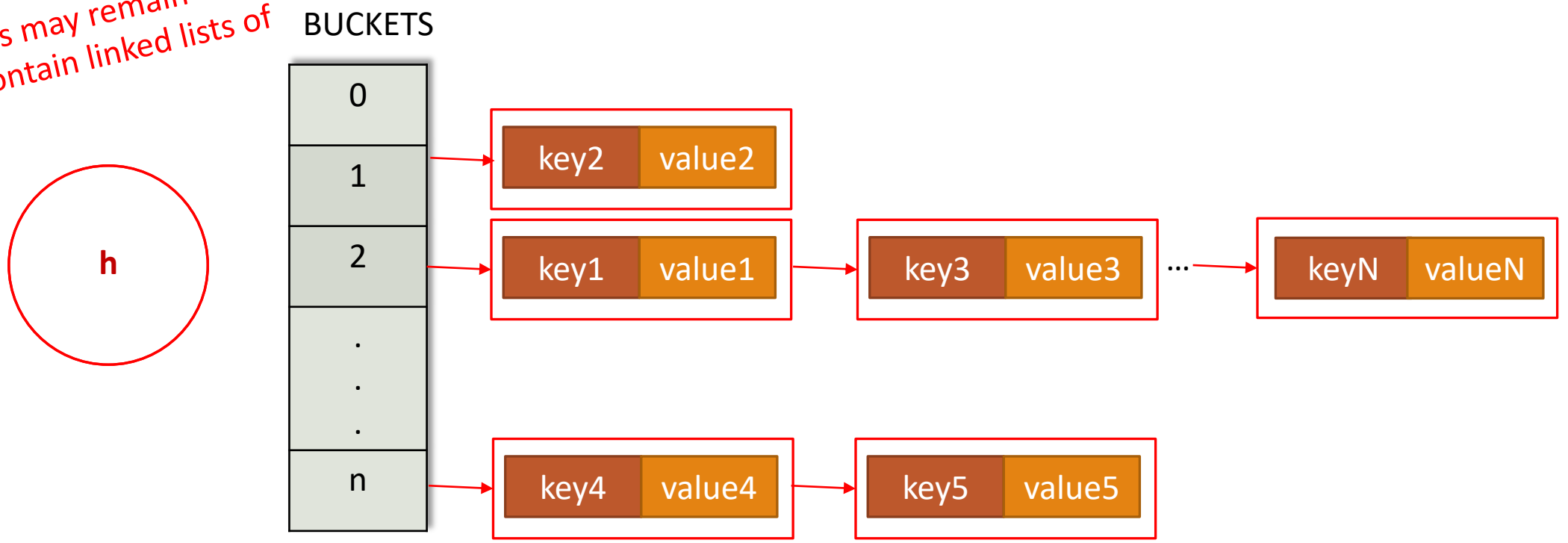


INTERNAL STRUCTURE

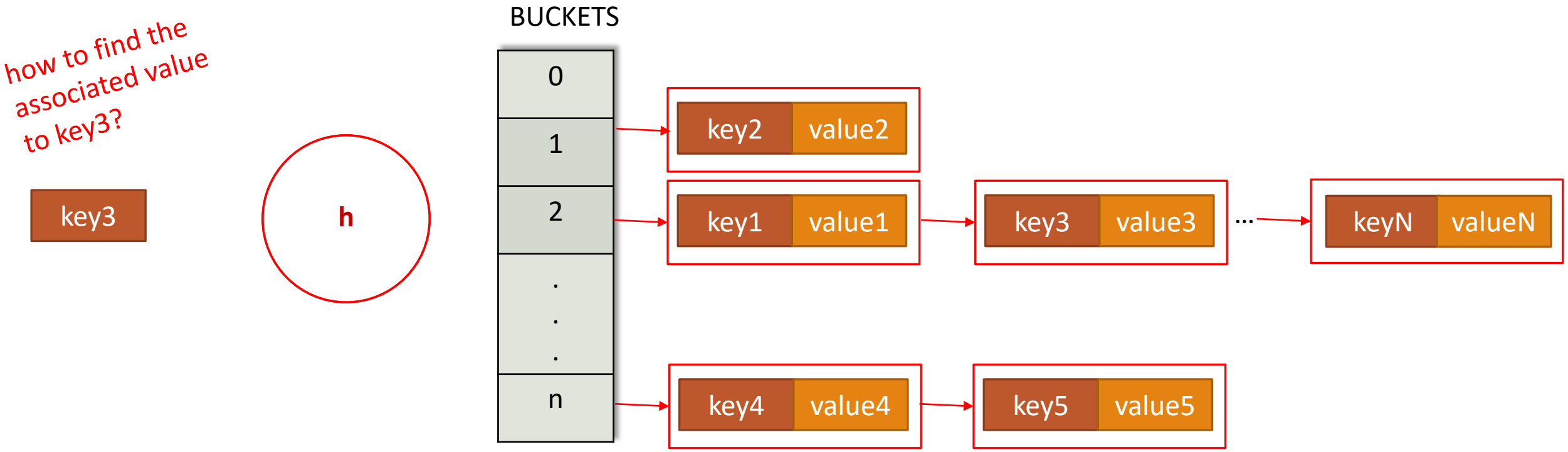


INTERNAL STRUCTURE

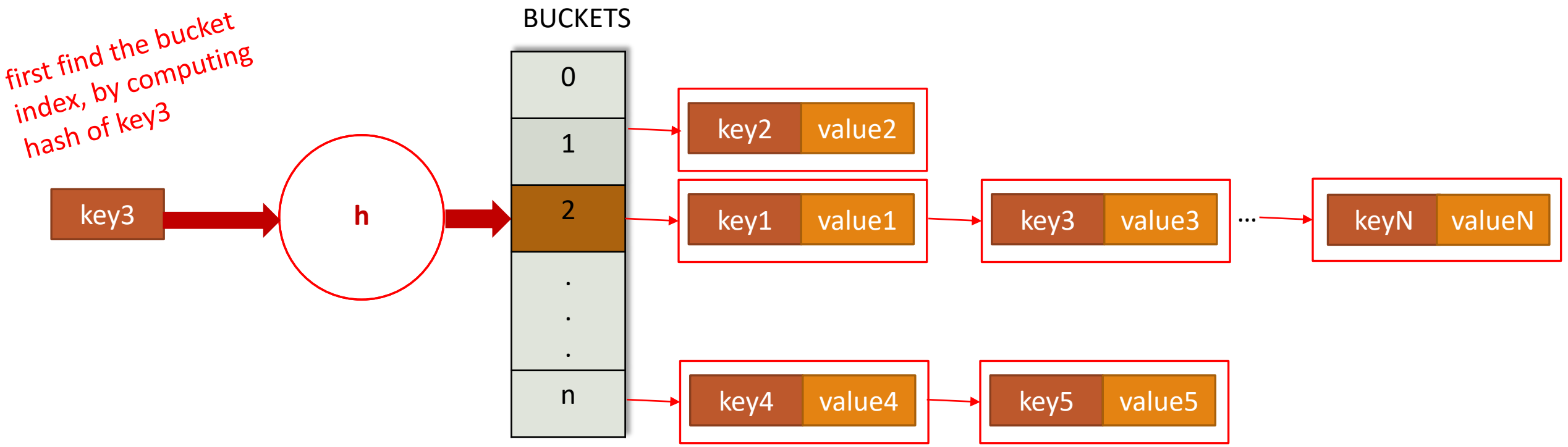
finally, some buckets may remain empty and some others contain linked lists of key-value pairs



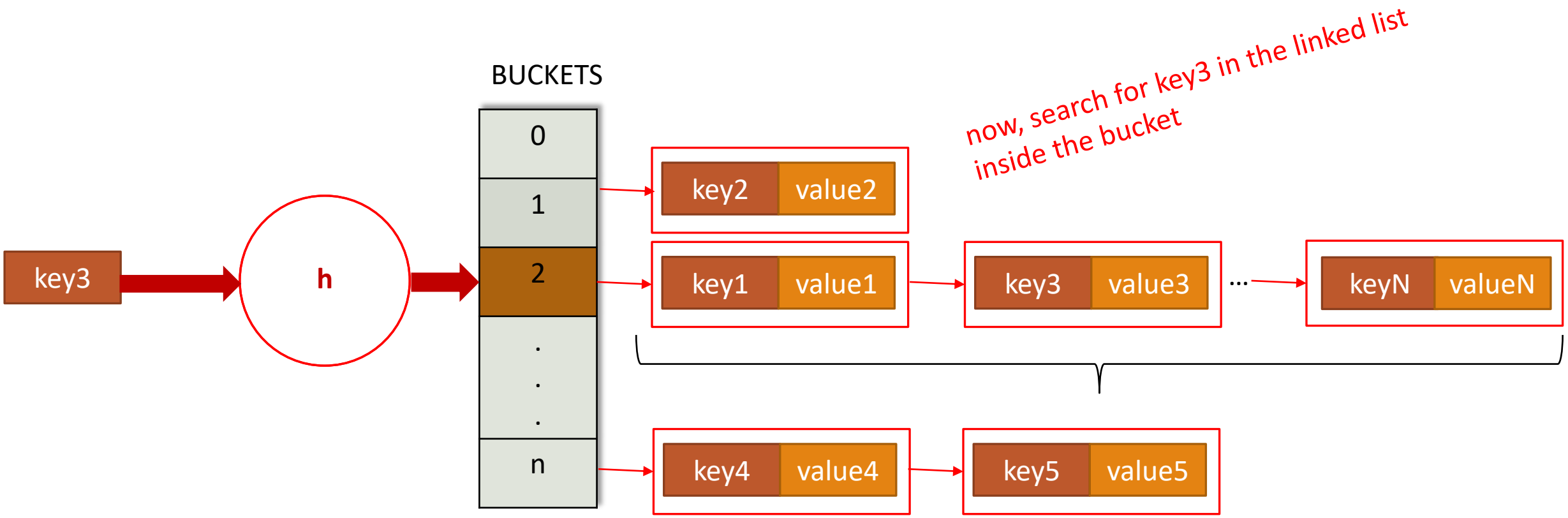
RETRIEVING VALUE for a KEY



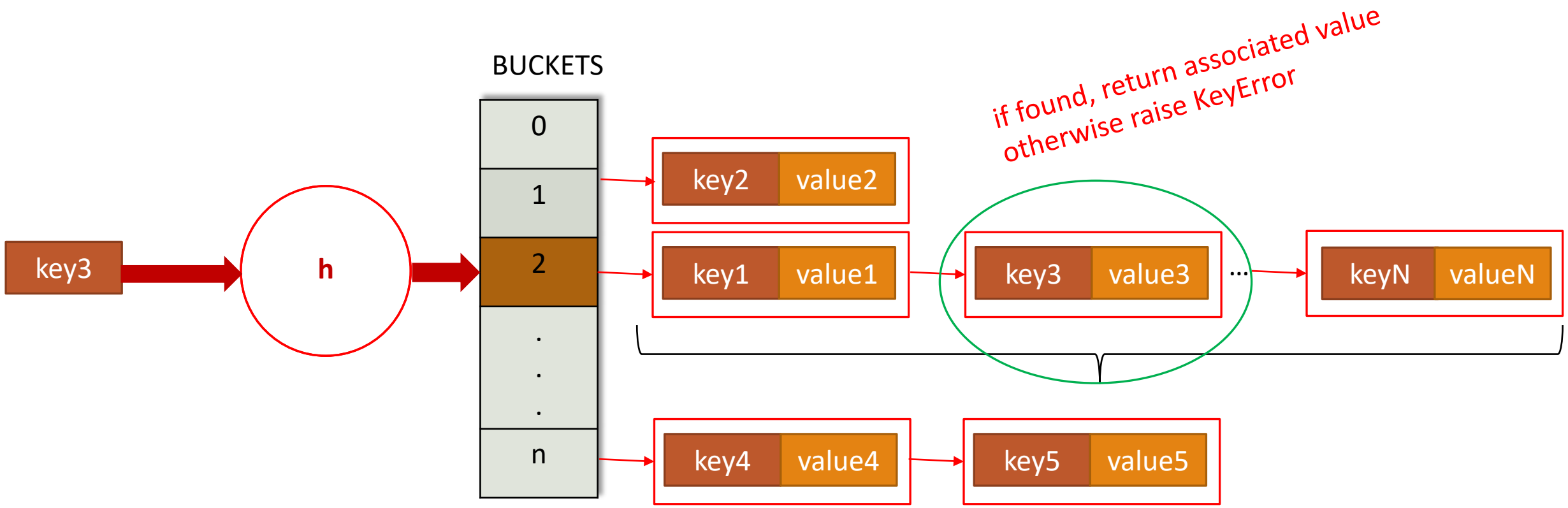
RETRIEVING VALUE for a KEY



RETRIEVING VALUE for a KEY



RETRIEVING VALUE for a KEY



HASH FUNCTION

- the idea of hashing is to distribute the entries (key/value pairs) across an array of **buckets**
- a good hash function is a function which results in less **collisions**
- a critical statistic for a hash table is the *load factor*, defined as

$$load\ factor = \frac{n}{k}$$

where
  - *n* is the number of entries occupied in the hash table.
  - *k* is the number of buckets.
- as the load factor grows larger, the hash table becomes slower, and it may even fail to work
- the expected constant time property of a hash table assumes that the load factor be kept below some bound