Dictionary and Character Encoding

HADAIQ ROLIS SANABILA

What is dictionary?

- In data structure terms, a dictionary is better termed an associative array, associative list or a map
- You can think of it as a list of pairs, where the first element of the pair, the key, is used to retrieve the second element, the value
- Thus we map a key to a value

Key-Value Pairs

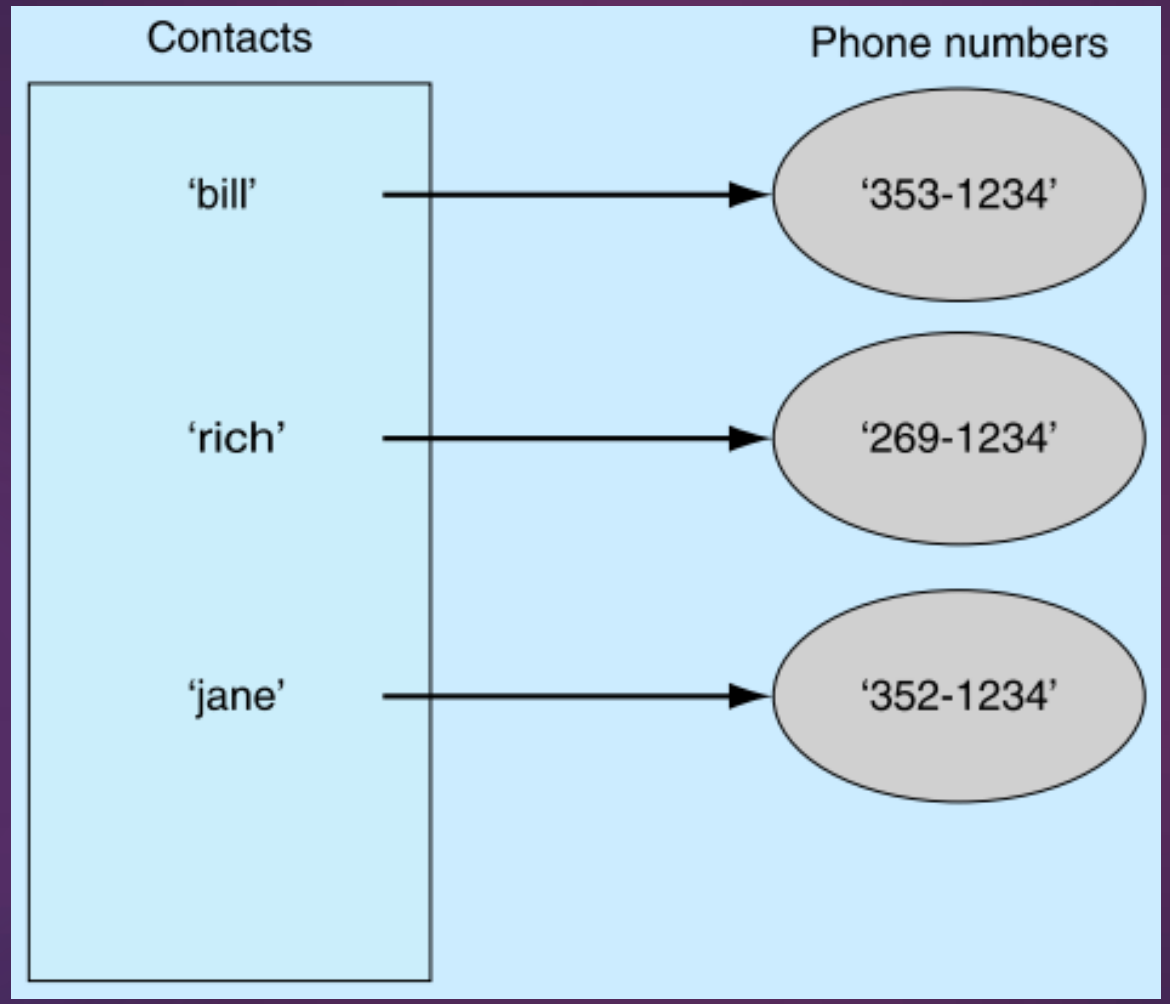
- The key acts as an index to find the associated value
- Just like a dictionary, you look up a word by its spelling to find the associated definition
- A dictionary can be searched to locate the value associated with a key

Python Dictionary

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Use { } marker to create a dictionary
- Use the : marker to indicate key : value pairs

```
contacts = {'bill':'353-1234', 'rich':'269-1234', 'jane':'352-1234'}
print (contacts)
{'bill':'353-1234',
'rich':'269-1234',
'jane':'352-1234'}
```



Empty Dictionary

- A pair of curly brackets creates an empty dictionary
- Alternately, one can use `dict` with no argument

```
some_dictio = { }
another_diction = dict()
```

- **Key** must be **immutable**
  - String, int, tuples are fine
  - Lists are not
- **Value** can be **anything**

Collections but not a sequence

- Dictionaries are collections but they are not sequences such as lists, strings, or tuples
  - There is no order to the elements of a dictionary
  - In fact, the order (for example, when printed) might change as elements are added or deleted
- So how to access dictionary elements?

Accessing dictionary elements

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Access require [ ], and the key is the index

```
>>> my_diction = {}
    #empty dict
>>> my_diction ['bill'] =25
    #assign the pair 'bill':25
>>> print (my_diction['bill'])
    #print 25
```

Dictionaries are mutable

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Like list, dictionaries are mutable data structure
  - You can change the object via various operations, such as index assignment

```
my_diction = {'Bill':2, 'Rich':10}
print (my_diction['Bill']) # print 2
My_diction ['Bill'] = 100
print (my_diction['Bill']) # print 100
```

Common operations

- `len (my_dictio)`
  - Number of key:value **pairs** in the dictionary
- `element in my_dictio`
  - Boolean: is element a **key** in the dictionary?
- `for key in my_diction:`
  - Iterates through the **keys** in the dictionary
- `my_diction.clear()`
  - Empty the dictionary
- `my_diction.copy()`
  - Shallow copy

Dictionary Method

Operation	Explanation
d.items()	Returns a view of the (key, value) pairs in d
d.keys()	Returns a view of the keys of d
d.pop(key)	Removes the (key, value) pair with key key from d and returns the value
d.update(d2)	Adds the (key, value) pairs of dictionary d2 to d
d.values()	Returns a view of the values of d

The containers returned by `d.items()`, `d.keys()`, and `d.values()` (called views) can be iterated over

```
>>> days
{'Mo': 1, 'Tu': 2, 'Th': 4, 'W': 3}
>>> days.pop('Tu')
2
>>> days
{'Mo': 1, 'Th': 4, 'W': 3}
>>> days2 = {'Tu':2, 'Fr':5}
>>> days.update(days2)
>>> days
{'Fr': 5, 'W': 3, 'Th': 4, 'Mo': 1,
'Tu': 2}
>>> days.items()
dict_items([('Fr', 5), ('W', 3), ('Th',
4), ('Mo', 1), ('Tu', 2)])
>>> days.keys()
dict_keys(['Fr', 'W', 'Th', 'Mo', 'Tu'])
>>> vals = days.values()
>>> vals
dict_values([5, 3, 4, 1, 2])
>>> for val in vals:
        print(val, end=' ')


5 3 4 1 2
>>>
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

Dictionary and multiway if statement

Uses of a dictionary:

- container with custom indexes
- alternative to the multi-way `if` statement

```
def complete(abbreviation):
   #'returns day of the week corresponding to abbreviation'
    if abbreviation == 'Mo':
        return 'Monday'
    elif abbreviation == 'Tu':
        return 'Tuesday'
    elif
        ......
    else: # abbreviation must be Su
        return 'Sunday'
```

```
def complete(abbreviation):
    'returns day of the week corresponding to abbreviation'
    days = {'Mo': 'Monday', 'Tu':'Tuesday', 'We': 'Wednesday',
            'Th': 'Thursday', 'Fr': 'Friday', 'Sa': 'Saturday',
            'Su':'Sunday'}
    return days[abbreviation]
```

Dictionary as a container of counters

Problem: computing the number of occurrences of items in a list

```
>>> grades = [95, 96, 100, 85, 95, 90, 95, 100, 100]
>>> frequency(grades)
{96: 1, 90: 1, 100: 3, 85: 1, 95: 3}
>>>
```

Solution: Iterate through the list and, for each grade, increment the counter corresponding to the grade.

Problems:

- impossible to create counters before seeing what's in the list
- how to store grade counters so a counter is accessible using the corresponding grade

Solution: a dictionary mapping a grade (the key) to its counter (the value)

Dictionary as a container of counters

Problem: computing the number of occurrences of items in a list

```
>>> grades = [95, 96, 100, 85, 95, 90, 95, 100, 100]
              ^   ^    ^    ^   ^   ^   ^
```

counters

95	96	100	85	90
3	1	1	1	1

```
def frequency(itemList):
    'returns frequency of items in itemList'

    counters = {}
    for item in itemList:
        if item in counters: # increment item counter
            counters[item] += 1
        else: # create item counter
            counters[item] = 1
    return counters
```

Exercise

Implement function `wordcount()` that takes as input a text—as a string—and prints the frequency of each word in the text; assume there is no punctuation in the text.

```
>>> text = 'all animals are equal but some animals are more equal than other'
>>> wordCount(text)
all       appears 1 time.
animals   appears 2 times.
some      appears 1 time.
equal     appears 2 times.
but       appears 1 time.
other     appears 1 time.
are       appears 2 times.
than      appears 1 time.
more      appears 1 time.
>>>
```

Exercise

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

```
def wordCount(text):
    wordList = text.split()  # split text into list of words

    counters ={}             # dictionary of counters
    for word in wordList:
        if word in counters: # counter for word exists
            counters[word] += 1
        else:                # counter for word doesn't exist
            counters[word] = 1

    for word in counters:    # print word counts
        if counters[word] == 1:
            print('{:8} appears {} time.'.format(word, counters[word]))
        else:
            print('{:8} appears {} times.'.format(word, counters[word]))
```

Exercise

Implement function `lookup()` that implements a phone book lookup application. Your function takes, as input, a dictionary representing a phone book, mappingtuples (containing the first and last name) to strings (containing phone numbers)

```
>>> phonebook = {
        ('Anna','Karenina'):'(123)456-78-90',
        ('Yu', 'Tsun'):'(901)234-56-78',
        ('Hans', 'Castorp'):'(321)908-76-54'}
>>> lookup(phonebook)
Enter the first name: Anna
Enter the last name: Karenina
(123)456-78-90
```

```
def lookup(phonebook):
    '''implements interactive phone book service us
       phonebook dictionary'''
        first = input('Enter the first name: ')
        last = input('Enter the last name: ')

        person = (first, last)    # construct the key

        if person in phonebook:   # if key is in dictionary
            print(phonebook[person])  # print value
        else:                     # if key not in dictionary
            print('The name you entered is not known.')
```

Character Encoding

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- A string (str) object contains an ordered sequence of characters which can be any of the following:
  - lowercase and uppercase letters in the English alphabet: a b c … z and A B C … Z
  - decimal digits: 0 1 2 3 4 5 6 7 8 9
  - punctuation: , . : ; ‘ “ ! ? etc.
  - Mathematical operators and common symbols: = < > + - / * $ # % @ & etc.
  - More later
- Each character is mapped to a specific bit encoding, and this encoding maps back to the character.
- For many years, the standard encoding for characters in the English language was the American Standard Code for Information Interchange (ASCII)

ASCII

32		48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		

- For many years, the standard encoding for characters in the English language was the American Standard Code for Information Interchange (ASCII)
- The code for a is 97, which is 01100001 in binary or 0x61 in hexadecimal notation
- The encoding for each ASCII character fits in 1 byte (8 bits)

Built in function `ord()` and `char()`

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

```
>>> ord('a')
97
>>> ord('?')
63
>>> ord('\n')
10
>>> chr(10)
'\n'
>>> chr(63)
'?'
>>> chr(97)
'a'
>>>
```

Function `ord()` takes a character (i.e., a string of length 1) as input and returns its ASCII code

Function `chr()` takes an ASCII encoding (i.e., a non-negative integer) and returns the corresponding character

Beyond ASCII

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- A string object contains an ordered sequence of characters which can be any of the following:
  - lowercase and uppercase letters in the English alphabet: a … z and A … Z
  - decimal digits: 0 1 2 3 4 5 6 7 8 9
  - punctuation: , . : ; ‘ “ ! ? etc.
  - Mathematical operators and common symbols: = < > + - / * $ # % @ & etc.
  - Characters from languages other than English
  - Technical symbols from math, science, engineering, etc.
- There are only 128 characters in the ASCII encoding
- Unicode has been developed to be the universal character encoding scheme

Unicode

- In Unicode, every character is represented by an integer code point.
  - The code point is not necessarily the actual byte representation of the character; it is just the identifier for the particular character
- The code point for letter a is the integer with hexadecimal value 0x0061
  - Unicode conveniently uses a code point for ASCII characters that is equal to their ASCII code

escape sequence \u indicates start of Unicode code point

```
>>> '\u0061'
'a'
>>> '\u0064\u0061d'
'dad'
>>>
'\u0409\u0443\u0431\u043e\u043c\u
0438\u0440'
'Љубомир'
>>> '\u4e16\u754c\u60a8\u597d!'
'世界您好!'
>>>
```

Comprehesion List

- The use of lists in Python is a major part of its power
- Lists are very useful and can be used to accomplish many task
- Therefore, Python provides some pretty powerful support to make common list tasks easier

List Comprehesion

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Format:

  [<expression> <for clause> <condition>]

- Example

```
>>> [ i for i in range (20) if i%2 == 0]
    [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

- Modifying what we collect

```
>>> [n**2 for n in range (1,6)]
    [1, 4, 9, 16, 25]
```

Multiple collects


```
[x+y for x in range (1,4) for y in range (1,4)]
```

It is as if we had done the following :

```
my_list = []
for x in range (1,4):
    for y in range (1,4):
        my_list.append(x+y)
```

→ [2,3,4,3,4,5, 4, 5, 6]