Stacks, Queues, and Symbol Tables
Outline

1 Stacks

2 Queues

3 Symbol Tables
Stacks

A stack is an iterable collection that is based on the last-in-first-out (LIFO) policy. It is an iterable data type.

ArrayStack

ArrayStack()
initialize an empty stack

s

s.isEmpty()
is empty?

len(s)
number of elements in

s.push(item)
push item on top of s

s.peek()
peek and return item on top of s

s.pop()
pop and return the item on top of s

iter(s)
an iterator over the elements of s
A stack is an iterable collection that is based on the last-in-first-out (LIFO) policy.
A stack is an iterable collection that is based on the last-in-first-out (LIFO) policy.

An iterable data type `ArrayStack` that represents a stack:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArrayStack()</code></td>
<td>Initialize an empty stack ( s )</td>
</tr>
<tr>
<td><code>s.isEmpty()</code></td>
<td>Is ( s ) empty?</td>
</tr>
<tr>
<td><code>len(s)</code></td>
<td>Number of elements in ( s )</td>
</tr>
<tr>
<td><code>s.push(item)</code></td>
<td>Push ( item ) on top of ( s )</td>
</tr>
<tr>
<td><code>s.peek()</code></td>
<td>Peek and return ( item ) on top of ( s )</td>
</tr>
<tr>
<td><code>s.pop()</code></td>
<td>Pop and return the item on top of ( s )</td>
</tr>
<tr>
<td><code>iter(s)</code></td>
<td>An iterator over the elements of ( s )</td>
</tr>
</tbody>
</table>
Stacks

Program:
reverse.py

• Standard input: a sequence of strings
• Standard output: the strings in reverse order

/terminal
~/workspace/ipp/programs
$
python3 reverse .py
b o l t o n
<ctrl -d>
n o t l o b
Program: reverse.py

Stacks
Program: reverse.py

- Standard input: a sequence of strings
Program: reverse.py

- Standard input: a sequence of strings
- Standard output: the strings in reverse order
Program: reverse.py

- Standard input: a sequence of strings
- Standard output: the strings in reverse order

```bash
$ python3 reverse.py
b o l t o n
<ctrl-d>
n o t l o b
```
from arraystack import ArrayStack
import stdio

def main():
    stack = ArrayStack()
    while not stdio.isEmpty():
        s = stdio.readString()
        stack.push(s)
    for s in stack:
        stdio.write(s + ' ')
    stdio.writeln()

if __name__ == '__main__':
    main()
from arraystack import ArrayStack
import stdio

def main():
    stack = ArrayStack()
    while not stdio.isEmpty():
        s = stdio.readString()
        stack.push(s)
    for s in stack:
        stdio.write(s + ' ')
    stdio.writeln()

if __name__ == '__main__':
    main()
import stdio

class ArrayStack:
    def __init__(self):
        self._a = []
    def isEmpty(self):
        return len(self) == 0
    def __len__(self):
        return len(self._a)
    def push(self, item):
        self._a.append(item)
    def peek(self):
        if self.isEmpty():
            raise Exception('Stack underflow')
        return self._a[-1]
    def pop(self):
        if self.isEmpty():
            raise Exception('Stack underflow')
        return self._a.pop(-1)
    def __iter__(self):
        return iter(reversed(self._a))

def _main():
    stack = ArrayStack()
    while not stdio.isEmpty():
        item = stdio.readString()
        if item != '-':
            stack.push(item)
        elif not stack.isEmpty():
import stdio

class ArrayStack:
    def __init__(self):
        self._a = []
    
def isEmpty(self):
        return len(self) == 0
    
def __len__(self):
        return len(self._a)
    
def push(self, item):
        self._a.append(item)
    
def peek(self):
        if self.isEmpty():
            raise Exception('Stack underflow')
        return self._a[-1]
    
def pop(self):
        if self.isEmpty():
            raise Exception('Stack underflow')
        return self._a.pop(-1)
    
def __iter__(self):
        return iter(reversed(self._a))

def _main():
    stack = ArrayStack()
    while not stdio.isEmpty():
        item = stdio.readString()
        if item != '-':
            stack.push(item)
        elif not stack.isEmpty():
            stack.pop()
```
stdio.write(str(stack.pop()) + ', ')
stdio.writeln('(' + str(len(stack)) + ' left on stack')

if __name__ == '__main__':
    __main__
```
Queues

A queue is an iterable collection that is based on the first-in-first-out (FIFO) policy.

An iterable data type `ArrayQueue` that represents a queue.

```python
ArrayQueue()
```

Initialize an empty queue.

```python
q
```

```python
q.isEmpty()
```

Is `q` empty?

```python
len(q)
```

Number of elements in `q`.

```python
q.enqueue(item)
```

Add `item` to the end of `q`.

```python
q.peek()
```

Peek and return the first item of `q`.

```python
q.dequeue()
```

Remove and return the first item of `q`.

```python
iter(q)
```

An iterator over the elements of `q`. 

```python
```
A queue is an iterable collection that is based on the first-in-first-out (FIFO) policy
A queue is an iterable collection that is based on the first-in-first-out (FIFO) policy

An iterable data type `ArrayQueue` that represents a queue

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ArrayQueue()</code></td>
<td>Initialize an empty queue <code>q</code></td>
</tr>
<tr>
<td><code>q.isEmpty()</code></td>
<td>Is <code>q</code> empty</td>
</tr>
<tr>
<td><code>len(q)</code></td>
<td>Number of elements in <code>q</code></td>
</tr>
<tr>
<td><code>q.enqueue(item)</code></td>
<td>Add <code>item</code> to the end of <code>q</code></td>
</tr>
<tr>
<td><code>q.peek()</code></td>
<td>Peek and return the first item of <code>q</code></td>
</tr>
<tr>
<td><code>q.dequeue()</code></td>
<td>Remove and return the first item of <code>q</code></td>
</tr>
<tr>
<td><code>iter(q)</code></td>
<td>An iterator over the elements of <code>q</code></td>
</tr>
</tbody>
</table>
Program: kthfromlast.py

• Command-line input: k (int)

• Standard input: sequence of strings

• Standard output: k th string from the end

~/workspace/ipp/programs

$ python3 kthfromlast .py 5

she sells sea shells on the sea shore

<ctrl -d>

shells
Program: kthfromlast.py
Program: kthfromlast.py

- Command-line input: $k$ (int)
Program: kthfromlast.py

- Command-line input: \( k \) (int)
- Standard input: sequence of strings
Program: kthfromlast.py

- Command-line input: \( k \) (int)
- Standard input: sequence of strings
- Standard output: \( k \)th string from the end
Program: `kthfromlast.py`

- Command-line input: $k$ (int)
- Standard input: sequence of strings
- Standard output: $k$th string from the end

```
$ python3 kthfromlast.py 5
she sells sea shells on the sea shore
<ctrl-d>
sheells
```
Queues

```python
from arrayqueue import ArrayQueue
import stdio
import sys

def main():
    k = int(sys.argv[1])
    queue = ArrayQueue()
    while not stdio.isEmpty():
        s = stdio.readString()
        queue.enqueue(s)
    n = len(queue)
    for i in range(1, n - k + 1):
        queue.dequeue()
    stdio.writeln(queue.peek())

if __name__ == '__main__':
    main()
```

from arrayqueue import ArrayQueue
import stdio
import sys

def main():
    k = int(sys.argv[1])
    queue = ArrayQueue()
    while not stdio.isEmpty():
        s = stdio.readString()
        queue.enqueue(s)
        n = len(queue)
        for i in range(1, n - k + 1):
            queue.dequeue()
        stdio.writeln(queue.peek())

if __name__ == '__main__':
    main()
import stdio

class ArrayQueue:
    def __init__(self):
        self._a = []
    def isEmpty(self):
        return len(self) == 0
    def __len__(self):
        return len(self._a)
    def enqueue(self, item):
        self._a.append(item)
    def peek(self):
        if self.isEmpty():
            raise Exception('Queue underflow')
        return self._a[0]
    def dequeue(self):
        if self.isEmpty():
            raise Exception('Queue underflow')
        return self._a.pop(0)
    def __iter__(self):
        return iter(self._a)

def _main():
    queue = ArrayQueue()
    while not stdio.isEmpty():
        item = stdio.readString()
        if item != '-':
            queue.enqueue(item)
        elif not queue.isEmpty():

Queue
```python
import stdio

class ArrayQueue:
    def __init__(self):
        self._a = []

def isEmpty(self):
    return len(self) == 0

def __len__(self):
    return len(self._a)

def enqueue(self, item):
    self._a.append(item)

def peek(self):
    if self.isEmpty():
        raise Exception('Queue underflow')
    return self._a[0]

def dequeue(self):
    if self.isEmpty():
        raise Exception('Queue underflow')
    return self._a.pop(0)

def __iter__(self):
    return iter(self._a)

def _main():
    queue = ArrayQueue()
    while not stdio.isEmpty():
        item = stdio.readString()
        if item != '-':
            queue.enqueue(item)
        elif not queue.isEmpty():
            print(queue.dequeue())
```

Queues

ArrayQueue.py

```python
stdio.write(str(queue.dequeue()) + '
')
stdio.writeln('(' + str(len(queue)) + ' left on queue')

if __name__ == '__main__':
    _main()
```
Symbol Tables

A symbol table is a data structure for key-value pairs that supports two operations: insert (put) a new pair into the table and search (get) the value associated with a given key. No duplicate keys are allowed; when we put a key-value pair into a table already containing that key (and an associated value), the new value replaces the old one.

Applications

<table>
<thead>
<tr>
<th>Application Purpose</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary</td>
<td>find</td>
<td>definition</td>
</tr>
<tr>
<td>book index</td>
<td>find</td>
<td>relevant pages</td>
</tr>
<tr>
<td>term</td>
<td>list of page numbers</td>
<td></td>
</tr>
<tr>
<td>web search</td>
<td>find</td>
<td>relevant web pages</td>
</tr>
<tr>
<td>keyword</td>
<td>list of page names</td>
<td></td>
</tr>
</tbody>
</table>
A symbol table is a data structure for key-value pairs that supports two operations: insert (put) a new pair into the table and search (get) the value associated with a given key.
Symbol Tables

A symbol table is a data structure for key-value pairs that supports two operations: insert (put) a new pair into the table and search (get) the value associated with a given key.

No duplicate keys are allowed; when we put a key-value pair into a table already containing that key (and an associated value), the new value replaces the old one.
A symbol table is a data structure for key-value pairs that supports two operations: insert (put) a new pair into the table and search (get) the value associated with a given key.

No duplicate keys are allowed; when we put a key-value pair into a table already containing that key (and an associated value), the new value replaces the old one.

Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Purpose</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary</td>
<td>find definition</td>
<td>word</td>
<td>definition</td>
</tr>
<tr>
<td>book index</td>
<td>find relevant pages</td>
<td>term</td>
<td>list of page numbers</td>
</tr>
<tr>
<td>web search</td>
<td>find relevant web pages</td>
<td>keyword</td>
<td>list of page names</td>
</tr>
</tbody>
</table>
Symbol Tables

SymbolTable()
constructs an empty symbol table

s.isEmpty()
returns True if s is empty, and False otherwise

len(s)
returns the number of key-value pairs in s

d in s
returns True if s contains d, and False otherwise

s[d]
returns the value associated with d in s

s[d] = val
inserts the pair d/val into s

s.keys()
returns the keys in s as an iterable object

s.values()
returns the values in s as an iterable object
## Symbol Tables

<table>
<thead>
<tr>
<th>SymbolTable()</th>
<th>constructs an empty symbol table $s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$.isEmpty()</td>
<td>returns True if $s$ is empty, and False otherwise</td>
</tr>
<tr>
<td>len($s$)</td>
<td>returns the number of key-value pairs in $s$</td>
</tr>
<tr>
<td>key in $s$</td>
<td>returns True if $s$ contains key, and False otherwise</td>
</tr>
<tr>
<td>$s$[key]</td>
<td>returns the value associated with key in $s$</td>
</tr>
<tr>
<td>$s$[key] = val</td>
<td>inserts the pair key/val into $s$</td>
</tr>
<tr>
<td>$s$.keys()</td>
<td>returns the keys in $s$ as an iterable object</td>
</tr>
<tr>
<td>$s$.values()</td>
<td>returns the values in $s$ as an iterable object</td>
</tr>
</tbody>
</table>
Symbol Tables

Program: frequencycounter.py

- Command-line input: minLen (int)
- Standard input: a sequence of words
- Standard output: for the words that are at least as long as minLen, writes the total word count, the number of distinct words, and the most frequent word

```
~/workspace/ipp/programs$
$ python3 frequencycounter.py 8 < ../data/tale.txt
Word count: 13525
Distinct word count: 4371
Most frequent word: business (134 repetitions)
```
Program: frequencycounter.py

Terminal:
```bash
$ python3 frequencycounter.py 8 < ../data/tale.txt
Word count: 13525
Distinct word count: 4371
Most frequent word: business (134 repetitions)
```
Symbol Tables

Program: frequencycounter.py

- Command-line input: minLen (int)

Terminal
$ python3 frequencycounter.py 8 < ../data/tale.txt

Word count: 13525
Distinct word count: 4371
Most frequent word: business (134 repetitions)
Program: frequencycounter.py

- Command-line input: \textit{minLen} (int)
- Standard input: a sequence of words

```bash
$ python3 frequencycounter.py 8 < ../data/tale.txt
Word count: 13525
Distinct word count: 4371
Most frequent word: business (134 repetitions)
```
Program: frequencycounter.py

- Command-line input: \texttt{minLen} (int)
- Standard input: a sequence of words
- Standard output: for the words that are at least as long as \texttt{minLen}, writes the total word count, the number of distinct words, and the most frequent word
Program: frequencycounter.py

- Command-line input: minLen (int)
- Standard input: a sequence of words
- Standard output: for the words that are at least as long as minLen, writes the total word count, the number of distinct words, and the most frequent word

```bash
> ~/workspace/ipp/programs
$ python3 frequencycounter.py 8 < ../data/tale.txt
Word count: 13525
Distinct word count: 4371
Most frequent word: business (134 repetitions)
$`
Symbol Tables

```python
from symboltable import SymbolTable
import stdio
import sys

def main():
    minLen = int(sys.argv[1])
    distinct, words = 0, 0
    st = SymbolTable()
    while not stdio.isEmpty():
        word = stdio.readString()
        if len(word) < minLen:
            continue
        words += 1
        if word in st:
            st[word] += 1
        else:
            st[word] = 1
        distinct += 1
    maxFreq = 0
    maxFreqWord = ''
    for word in st.keys():
        if st[word] > maxFreq:
            maxFreq = st[word]
            maxFreqWord = word
    stdio.writeln('Word count: ' + str(words))
    stdio.writeln('Distinct word count: ' + str(distinct))
    stdio.writef('Most frequent word: %s (%d repetitions)
', maxFreqWord, maxFreq)

if __name__ == '__main__':
    main()
```
```python
from symboltable import SymbolTable
import stdio
import sys

def main():
    minLen = int(sys.argv[1])
    distinct, words = 0, 0
    st = SymbolTable()
    while not stdio.isEmpty():
        word = stdio.readString()
        if len(word) < minLen:
            continue
        words += 1
        if word in st:
            st[word] += 1
        else:
            st[word] = 1
            distinct += 1
    maxFreq = 0
    maxFreqWord = ''
    for word in st.keys():
        if st[word] > maxFreq:
            maxFreq = st[word]
            maxFreqWord = word
    stdio.writeln('Word count : ' + str(words))
    stdio.writeln('Distinct word count : ' + str(distinct))
    stdio.writef('Most frequent word: %s (%d repetitions)
', maxFreqWord, maxFreq)

if __name__ == '__main__':
    main()
```

This code snippet creates a frequency counter. It reads words from standard input, excluding those shorter than a specified minimum length. It stores the count of each word in a symbol table and finds the word with the maximum frequency. It outputs the total word count, the number of distinct words, and the most frequent word with its repetition count.
Symbol Tables

A dictionary (an object of the built-in mapping type `dict`) is an unordered set of key-value pairs, with the requirement that the keys be unique.

The simplest way to create a dictionary is to place comma-separated key-value pairs (the key and value within a pair are separated by a colon) between matching curly brackets.

Example (days of the week):

```python
```

The `len()` function can be used to obtain the number of key-value pairs in a dictionary; in the above example, `len(dow)` returns `7`.

The comparison operator `in` can be used to check if a particular key exists in a dictionary; in the above example, `5 in dow` evaluates to `True`, whereas `42 in dow` evaluates to `False`. 
A dictionary (an object of the built-in mapping type `dict`) is an unordered set of key-value pairs, with the requirement that the keys be unique.
A dictionary (an object of the built-in mapping type `dict`) is an unordered set of key-value pairs, with the requirement that the keys be unique.

The simplest way to create a dictionary is to place comma-separated key-value pairs (the key and value within a pair are separated by a colon) between matching curly brackets.
A dictionary (an object of the built-in mapping type `dict`) is an unordered set of key-value pairs, with the requirement that the keys be unique.

The simplest way to create a dictionary is to place comma-separated key-value pairs (the key and value within a pair are separated by a colon) between matching curly brackets.

Example (days of the week)

```python
```
A dictionary (an object of the built-in mapping type \texttt{dict}) is an unordered set of key-value pairs, with the requirement that the keys be unique.

The simplest way to create a dictionary is to place comma-separated key-value pairs (the key and value within a pair are separated by a colon) between matching curly brackets.

Example (days of the week)

```python
```

The \texttt{len()} function can be used to obtain the number of key-value pairs in a dictionary; in the above example, \texttt{len(dow)} returns 7.
A dictionary (an object of the built-in mapping type `dict`) is an unordered set of key-value pairs, with the requirement that the keys be unique.

The simplest way to create a dictionary is to place comma-separated key-value pairs (the key and value within a pair are separated by a colon) between matching curly brackets.

Example (days of the week)

```python
```

The `len()` function can be used to obtain the number of key-value pairs in a dictionary; in the above example, `len(dow)` returns 7.

The comparison operator `in` can be used to check if a particular key exists in a dictionary; in the above example, `5 in dow` evaluates to `True`, whereas `42 in dow` evaluates to `False`. 
Symbol Tables

If \( d \) is a dictionary, then \( d[\text{key}] \) returns the value associated with \( \text{key} \), and raises \( \text{KeyError}() \) if the key doesn't exist in the dictionary; in the above example, \( d\text{ow}[5] \) returns \'(Fri\)\', whereas \( d\text{ow}[42] \) raises \( \text{KeyError}() \).

The following statement inserts the key-value pair \( \text{key}/\text{val} \) into a dictionary \( d \):

\[
d[\text{key}] = \text{val}
\]

Note that if \( \text{key} \) is already in \( d \), then its value is updated to \( \text{val} \).

Example (add/update \( d\text{ow} \))

\[
d\text{ow}[7] = 'Error '
d\text{ow}[5] = 'Friday '
\]

If \( d \) is a dictionary, then \( d.\text{keys()} \) and \( d.\text{values()} \) respectively return the keys and values in \( d \) as an iterable object.

Example (iterate over keys and values of \( d\text{ow} \))

\[
\text{for key in } d\text{ow}.\text{keys()}: \\
\quad \text{stdio.writeln (key + '->' + d\text{ow}[key])}
\text{for val in } d\text{ow}.\text{values()}: \\
\quad \text{stdio.writeln (val)}
\]
Symbol Tables

If $d$ is a dictionary, then $d[\text{key}]$ returns the value associated with $\text{key}$, and raises `KeyError()` if the key doesn't exist in the dictionary; in the above example, $\text{dow}[5]$ returns 'Fri', whereas $\text{dow}[42]$ raises `KeyError()`.
Symbol Tables

If \( d \) is a dictionary, then \( d[\text{key}] \) returns the value associated with \( \text{key} \), and raises `KeyError()` if the key doesn’t exist in the dictionary; in the above example, \( \text{dow}[5] \) returns ‘Fri’, whereas \( \text{dow}[42] \) raises `KeyError()`.

The following statement inserts the key-value pair \( \text{key}/\text{val} \) into a dictionary \( d \)

\[
d[\text{key}] = \text{val}
\]

Note that if \( \text{key} \) is already in \( d \), then its value is updated to \( \text{val} \).
If \( d \) is a dictionary, then \( d[\text{key}] \) returns the value associated with \( \text{key} \), and raises \( \text{KeyError}() \) if the key doesn't exist in the dictionary; in the above example, \( \text{dow}[5] \) returns 'Fri', whereas \( \text{dow}[42] \) raises \( \text{KeyError}() \).

The following statement inserts the key-value pair \( \text{key}/\text{val} \) into a dictionary \( d \):

\[
d[\text{key}] = \text{val}
\]

Note that if \( \text{key} \) is already in \( d \), then its value is updated to \( \text{val} \).

Example (add/update \( \text{dow} \))

\[
\text{dow}[7] = 'Error'
\text{dow}[5] = 'Friday'
\]
Symbol Tables

If \( d \) is a dictionary, then \( d[key] \) returns the value associated with \( key \), and raises \( KeyError() \) if the key doesn’t exist in the dictionary; in the above example, \( \text{dow}[5] \) returns \'Fri\', whereas \( \text{dow}[42] \) raises \( KeyError() \)

The following statement inserts the key-value pair \( \text{key/val} \) into a dictionary \( d \)

\[
d[\text{key}] = \text{val}
\]

Note that if \( \text{key} \) is already in \( d \), then its value is updated to \( \text{val} \)

Example (add/update \( \text{dow} \))

\[
\begin{align*}
\text{dow}[7] &= \text{'Error'} \\
\text{dow}[5] &= \text{'Friday'}
\end{align*}
\]

If \( d \) is a dictionary, then \( d\text{.keys()} \) and \( d\text{.values()} \) respectively return the keys and values in \( d \) as an iterable object
Symbol Tables

If \( d \) is a dictionary, then \( d[\text{key}] \) returns the value associated with key, and raises KeyError() if the key doesn't exist in the dictionary; in the above example, \( \text{dow}[5] \) returns 'Fri', whereas \( \text{dow}[42] \) raises KeyError().

The following statement inserts the key-value pair key/val into a dictionary \( d \)

\[
d[\text{key}] = \text{val}
\]

Note that if key is already in \( d \), then its value is updated to val.

Example (add/update \( \text{dow} \))

\[
\text{dow}[7] = 'Error'
\text{dow}[5] = 'Friday'
\]

If \( d \) is a dictionary, then \( d.keys() \) and \( d.values() \) respectively return the keys and values in \( d \) as an iterable object.

Example (iterate over keys and values of \( \text{dow} \))

\[
\text{for key in dow.keys():}
\quad \text{stdio.writeln(key + '->' + dow[key]}
\text{for val in dow.values():}
\quad \text{stdio.writeln(val)}
\]
Symbol Tables

```python
import stdio

class SymbolTable:
    def __init__(self):
        self._st = {}
    def isEmpty(self):
        return len(self._st) == 0
    def __len__(self):
        return len(self._st)
    def __contains__(self, key):
        return key in self._st
    def __getitem__(self, key):
        return self._st[key]
    def __setitem__(self, key, val):
        self._st[key] = val
    def keys(self):
        return iter(self._st.keys())
    def values(self):
        return iter(self._st.values())

def _main():
    st = SymbolTable()
    st['Gautama'] = 'Siddhartha'
    st['Darwin'] = 'Charles'
    st['Einstein'] = 'Albert'
    stdio.writeln(st['Gautama'])
    stdio.writeln(st['Darwin'])
    stdio.writeln(st['Einstein'])

_main()
```
Symbol Tables

```python
import stdio

class SymbolTable:
    def __init__(self):
        self._st = {}

    def isEmpty(self):
        return len(self._st) == 0

    def __len__(self):
        return len(self._st)

    def __contains__(self, key):
        return key in self._st

    def __getitem__(self, key):
        return self._st[key]

    def __setitem__(self, key, val):
        self._st[key] = val

    def keys(self):
        return iter(self._st.keys())

    def values(self):
        return iter(self._st.values())

def _main():
    st = SymbolTable()
    st['Gautama'] = 'Siddhartha'
    st['Darwin'] = 'Charles'
    st['Einstein'] = 'Albert'
    stdio.writeln(st['Gautama'])
    stdio.writeln(st['Darwin'])
    stdio.writeln(st['Einstein'])
```

if 'Einstein' in st:
    stdio.writeln('Einstein found')
else:
    stdio.writeln('Einstein not found')
if 'Newton' in st:
    stdio.writeln('Newton found')
else:
    stdio.writeln('Newton not found')
for key in st.keys():
    stdio.writeln(key + ': ' + str(key))
for value in st.values():
    stdio.writeln(value)

if __name__ == '__main__':
    _main()