More on decision structures

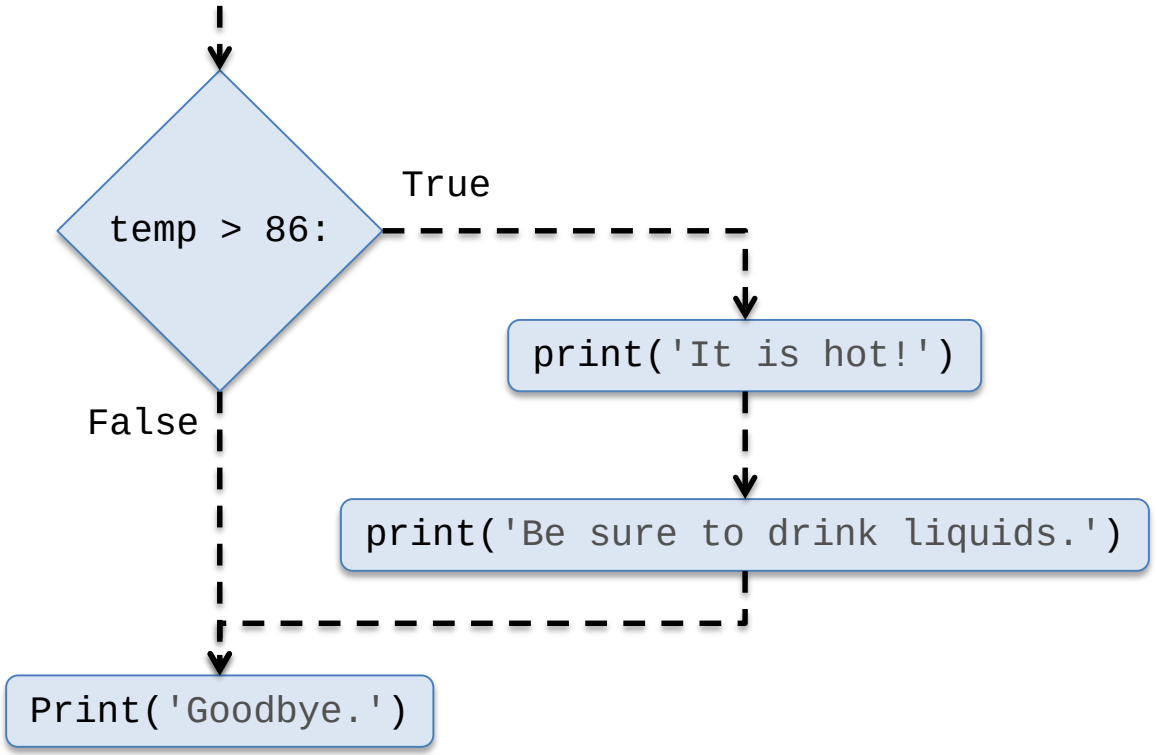
One-way, two-way and multi-way if statements

One-way if statement

```
if <condition>:
    <indented code block>
<non-indented statement>
```

```
if temp > 86:
    print('It is hot!')
    print('Be sure to drink liquids.')
print('Goodbye.')
```

The value of `temp` is 90.

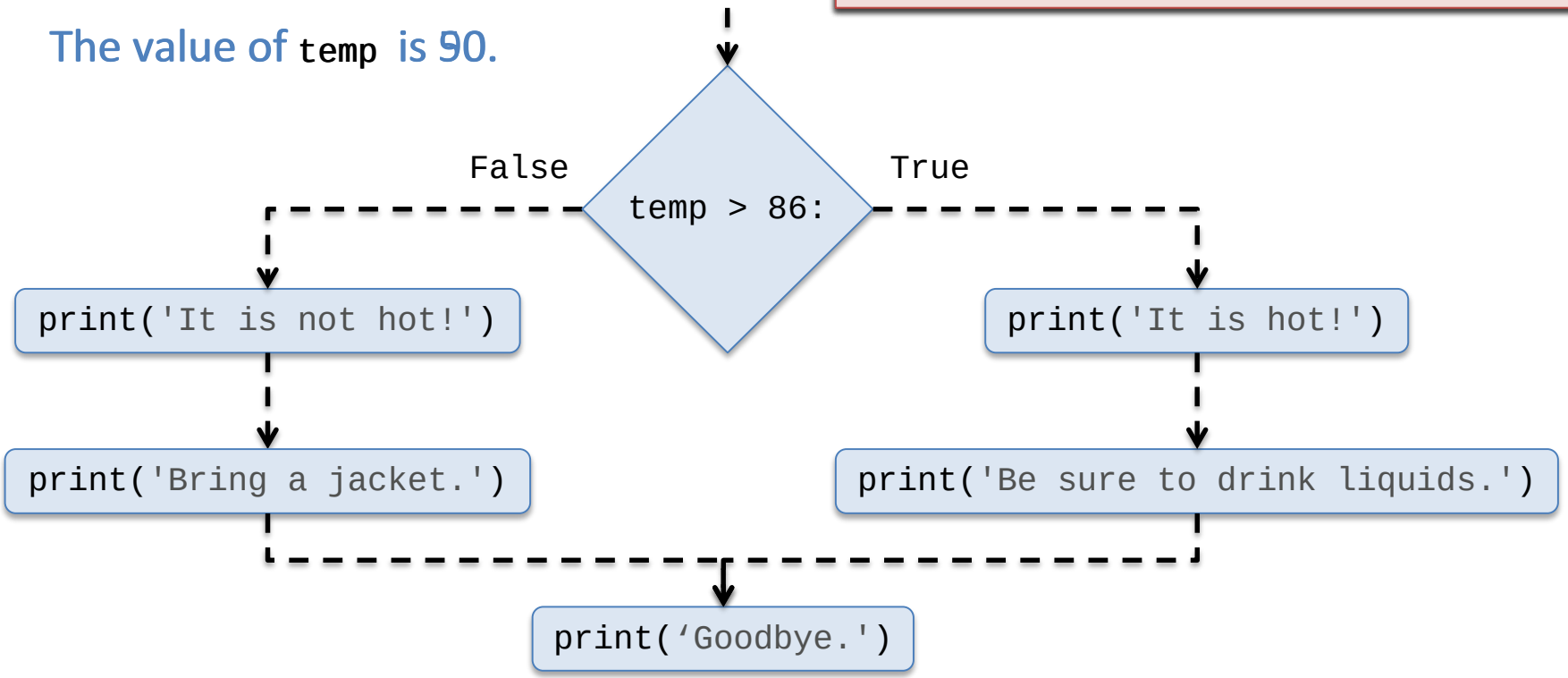


Two-way if statement

```
if <condition>:
    <indented code block 1>
else:
    <indented code block 2>
<non-indented statement>
```

```
if temp > 86:
    print('It is hot!')
    print('Be sure to drink liquids.')
else:
    print('It is not hot.')
    print('Bring a jacket.')
print('Goodbye.')
```

The value of `temp` is 90.



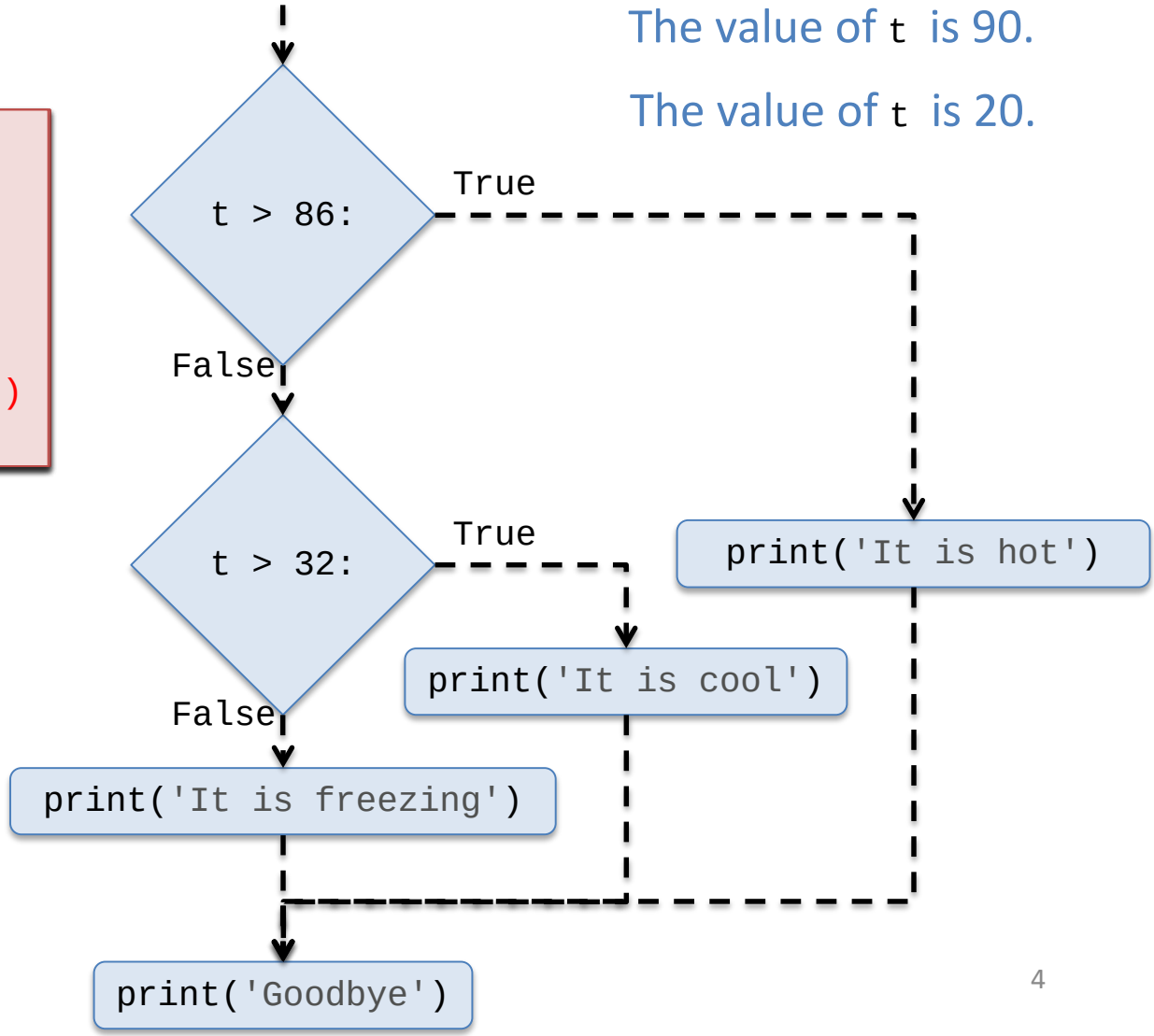
Multi-way if statement

```
def temperature(t):
    if t > 86:
        print('It is hot')
    elif t > 32:
        print('It is cool')
    else:
        print('It is freezing’)
    print('Goodbye')
```

The value of `t` is 50.

The value of `t` is 90.

The value of `t` is 20.



What is the wrong with this re-implementation of `temperature()`?

```
def temperature(t):
    if t > 32:
        print('It is cool')
    elif t > 86:
        print('It is hot')
    else: # t <= 32
        print('It is freezing')
    print('Goodbye')
```

```
def temperature(t):
    if 86 >= t > 32:
        print('It is cool')
    elif t > 86:
        print('It is hot')
    else: # t <= 32
        print('It is freezing')
    print('Goodbye')
```

The conditions must be
mutually exclusive,
either explicitly or implicitly

```
def temperature(t):
    if t > 86:
        print('It is hot')
    elif t > 32: # 86 >= t > 32
        print('It is cool')
    else: # t <= 32
        print('It is freezing')
    print('Goodbye')
```

More on iteration structures

Usage patterns

Iteration

The general format of a `for` loop statement is

```
for <variable> in <sequence>:
    <indented code block>
<non-indented code block>
```

`<indented code block>` is executed once for every item in `<sequence>`

- If `<sequence>` is a string then the items are its characters (each of which is a one-character string)
- If `<sequence>` is a list then the items are the objects in the list

`<non-indented code block>` is executed after every item in `<sequence>` has been processed

There are different `for` loop usage patterns

Accumulator Loop

Accumulating something in every loop iteration

Trace variables

x	s

Sum of numbers in a list

```
numList = [2,4,6,8,10,12]
s = 0
for x in numList:
    s += x
print (s)
```

Initialize accumulator variable to 0 outside of the loop

Accumulator Loop

Accumulating something in every loop iteration

Trace variables

x	c

Count the number of items in a list

```
numList = [2,4,6,8,10,12]
c = 0
for x in numList:
    c += 1
print (c)
```

Initialize accumulator variable to 0 outside of the loop

Accumulator Loop

Accumulating something in every loop iteration

Accumulate with a filter

Trace variables

x	s

Sum positive elements

```
numList = [2,4,6,-8,10,-12]
s = 0
for x in numList:
    if x > 0:
        s += x
print (s)
```

Initialize accumulator variable
to 0 outside of the loop

Accumulator Loop

Accumulating something in every loop iteration

Accumulate with a filter

Multiply numbers in a list

```
numList = [2,4,6,8,10,12]
m = 1
for x in numList:
    m *= x
print (m)
```

Trace variables

x	m

Initialize accumulator variable to 1 outside of the loop

Accumulator Loop

Accumulating something in every loop iteration

Trace variables

letter	accum

Accumulation loop with string

```
s = 'The Cow jumped Over the Moon'
accum = ''
for letter in s:
        if letter.isupper():
            accum += letter
 print (accum)
```

Initialize accumulator variable
to '' outside of the loop

Accumulator Loop

Accumulating something in every loop iteration

How often does string t occur in string s

```
s = 'The Cow jumped Over the Moon'
ct = 0
t = 'th'
for i in range(0, len(s) – len(t) + 1):
    if s[i:i+len(t)].lower() == t.lower():
        ct +=1
print (ct)
```

Initialize accumulator variable
to 0 outside of the loop

Trace variables

i	ct	s

Iterate through list/string…

Indexed loop

Test if list is sorted:

```
def is_sorted(lst):
    for i in range(len(lst) - 1):
        x = lst[i]
        y = lst[i + 1]
        if x > y:
            return False
    return True
```

Trace variables

i	x	y	x>y

Iterate through list/string…

Indexed loop

4-letter subwords of a word:

```
def subwords():
    s = 'abracadabra‘
    k = 4
    for i in range(0, len(s) - k + 1):
        print (s[i: i+k])
    return
```

Trace variables

i	s[i: i+k]

Iterate through list/string…

Indexed loop

Test if string is a palindrome:

```
def palindrome(s):
    for i in range(0, len(s)//2):
        if s[i] != s[-i-1]:
            return False
 return True
```

Trace variables

i	s[i]	s[-i-1]

Reading a file

Cursors

Reading a file

1. Open the file
2. Read from the file
3. Close the file.

Use built-in function to open

infile = open (filename, ‘r’)

where r is reading mode.

filename is the name of the file enclosed in quotes

What does “open for reading” mean?

- Upon opening for reading (in 'r' or default mode):
  - Python generates a “cursor”
  - The “cursor” points to the first character in the file
  - The “cursor” will then be moved by various methods of the “read family”
  - Again, each read method will MOVE the cursor
  - You need to pay attention to the position of the cursor!!

The 3 lines in this file end with the **new line character**

There is a blank line above this line

File methods

```
infile = open('example.txt')
```

Usage	Description
`infile.read(n)`	Read `n` characters starting from cursor; if fewer than `n` characters remain, read until the end of file. Returns a string of all read characters
`infile.read()`	Read starting from cursor up to the end of the file. Returns a string of all read characters
`infile.readline()`	Read starting from cursor up to, and including, the end of line character. Returns a string containing the read line.
`infile.readlines()`	Read starting from cursor up to the end of the file and return list of lines as strings
`outfile.write(s)`	Write string `s` to file `outfile` starting from cursor
`infile.close(n)`	Close file `infile`

- Methods `read()` and `readline()` return the characters read as a string
- Methods `readlines()` returns the characters read as a list of lines
- Method `write()` returns the number of characters written

Reading a file

```
1 The 3 lines in this file end with the new line character.\n
2 \n
3 There is a blank line above this line.\n
```

example.txt

When the file is opened, a cursor is associated with the opened file

The initial position of the cursor is:

- at the beginning of the file, if file mode is r

```
>>> infile = open('example.txt')
>>> infile.read(1)
'T'
>>> infile.read(5)
'he 3 '
>>> infile.readline()
'lines in this file end with the new line
character.\n'
>>> infile.read()
'\nThere is a blank line above this line.\n'
>>> infile.close()
>>>
```

Patterns for reading a text file

Common patterns for reading a file:

1. Read the file content into a string

Example:

```
def numChars(filename):
    'returns the number of characters in file filename'

    infile = open(filename, 'r')
    content = infile.read()
    infile.close()

    return len(content)
```

Patterns for reading a text file

Common patterns for reading a file:

1. Read the file content into a string
2. Read the file content into a list of words

Example:

```
def numWords(filename):
    'returns the number of words in file filename'

    infile = open(filename)
    content = infile.read()
    infile.close()
    wordList = content.split()

    return len(wordList)
```

Patterns for reading a text file

Common patterns for reading a file:

1. Read the file content into a string
2. Read the file content into a list of words
3. Read the file content into a list of lines

Example:

```
def numLines(filename):
    'returns the number of lines in file filename'

    infile = open(filename, 'r’)
    lineList = infile.readlines()
    infile.close()

    return len(lineList)
```

Iterating on lines of a file

- When files are sizable, it is inconvenient to store the entire content as a string in memory
- Instead it is more efficient to process one line of the file at a time

```
inFile = open(fname, 'r')
for line in inFile:
    DO STUFF TO line
inFile.close()
```

Examples

Example 1

- **Problem:** Write a function that finds the user-specified name and outputs the name and GMAT score
- Create a search algorithm
  1. Get the file – list of lines
  2. Two items per line, first item is name, second is score
  3. Use a For (list) to search the file for the specified name
  4. Output the name and score
- **Key Knowledge:**
  - Examine file contents
  - Read a line at a time to a list
  - In the For (list): compare the given name and the name on each list line

Example 2

- **Problem:** Write a function the determines the number of names that begin with the letter specified by the user.
- Create a search and count algorithm
  1. Get the file – list of lines
  2. Initialize a count variable
  3. Use the For (list) to compare the first letter of each name to the requested letter; if criteria is met, increment counter by 1
  4. Output count
- **Key Knowledge**
  - Choose lower() or upper() for letter comparison

Writing to a File

Writing to a text file

outfile = open('nameOfFile.txt', 'w')

- File must be open in writing mode
- If file does **not exist**, the open command CREATES one
- If file **exists**, the file is wiped clean
- Upon opening, a **cursor** is created that points to the first character of the file
- One then calls the **write** method on the outfile to write to file and move the cursor
- When interpreted, write will return the number of characters written to file

Writing to a text file

```
1 This is the first line. Still the first line…\n
2 Now we are in the second line.\n
3 Non string value like 5 must be converted first.\n
4 Non string value like 5 must be converted first.\n
```

test.txt

```
>>> outfile = open('test.txt', 'w')
>>> outfile.write('T')
1
>>> outfile.write('his is the first line.')
22
>>> outfile.write(' Still the first line...\n')
25
>>> outfile.write('Now we are in the second line.\n')
31
>>> outfile.write('Non string value like '+str(5)+' must be converted first.\n')
49
>>> outfile.write('Non string value like {} must be converted first.\n'.format(5))
49
>>> outfile.close()
```

Writing and flushing

outfile.flush()

- The open function in write mode creates a “buffer” in temporary memory
- The write method does not really write to hard disk but to the buffer
- Upon close, the actual writing to hard disk occurs
- Hence closing a file in w mode is essential
- One can also “flush” the buffer and force writing to hard disk while the file is still open

Example 1

- Problem: Write to a file named test.txt the following lines:

  I hear and I forget.

  I see and I remember.

  I do and I understand.

  [Confucius]

- Check that the file has been created
- Check that the writing took place

Example 2

- Problem: Retrieve all 3-letter words in test.txt and write to a new file
  1. Create a file to write to
  2. Get the file to read
  3. Read to the end of the file-returns a string of all characters read
  4. Convert the contents to a list of words
  5. In loop list:
     1. Test for a word with 3-characters
     2. Write to the new file