CS A131: Lecture 9

Nadia Ahmed

Orange Coast College, Computer Science

CS A131

Lecture 9: Overview

- File I/O
  - Processing records
  - Exceptions
- Lists and Tuples
  - Introduction
  - Slicing
  - Searching lists
  - List methods and built in functions
  - Copying
  - Processing
  - 2D lists
  - Tuples
  - Plots

Writing to a file: `add_coffee_record.py`

```python
def main():
  another = 'y'
  coffee_file = open('coffee.txt', 'a')
  while another == 'y' or another == 'Y':
    print('Enter the following coffee data:')
    descr = input('Description: ')
    qty = int(input('Quantity (in pounds): '))
    coffee_file.write(descr + '\n')
    coffee_file.write(str(qty) + '\n')

    print('Do you want to add another record?')
    another = input ('y/Y for yes, anything else for no: ')
  coffee_file.close()
  print('Data appended to coffee.txt')

main()
```

Reading a file: `read_coffee_record.py`

```python
def main():
  coffee_file = open('coffee.txt', 'r')
  descr = coffee_file.readline()

  while descr != '':
    qty = float(coffee_file.readline())
    descr = descr.rstrip('\n')
    print('Description: %s' % descr)
    print('Quantity: %f' % qty)

    descr = coffee_file.readline()
  coffee_file.close()

main()
```

Writing to a file: `search_coffee_record.py`

```python
def main():
  found = False
  search = input('Enter a description to search for: ')
  coffee_file = open('coffee.txt', 'r')

  descr = coffee_file.readline()

  while descr != '':
    qty = float(coffee_file.readline())
    descr = descr.rstrip('\n')
    if descr == search:
      print('Description: %s' % descr)
      print('Quantity: %f' % qty)
      print()
      found = True
    descr = coffee_file.readline()

  coffee_file.close()
  if not found:
    print('That item was not found in the file.')

main()
```

Modifying a file: `modify_coffee_record.py`

```python
import os
def main():
  found = False
  search = input('Enter a description to search for: ')
  new_qty = int(input('Enter the new quantity: '))
  coffee_file = open('coffee.txt', 'r')
  temp_file = open('temp.txt', 'w')
  descr = coffee_file.readline()
  while descr != '':
    qty = float(coffee_file.readline())
    descr = descr.rstrip('\n')
    if descr == search:
      temp_file.write(descr + '\n')
      temp_file.write(str(new_qty)) + '\n')
      found = True
      ...
```

Modifying a file: `modify_coffee_record.py`

```python
      ...
    else:
      temp_file.write(descr + '\n')
      temp_file.write(str(qty) + '\n')
    descr = coffee_file.readline()
  coffee_file.close()
  temp_file.close()
  os.remove('coffee.txt')
  os.rename('temp.txt', 'coffee.txt')
  if found:
    print('The file has been updated.')
  else:
    print('That item was not found in the file.')
main()
```

Exceptions

- An exception is an error that occurs while a program is running, causing the program to stop.
- You can use the `try/except` statement to handle exceptions.
- Use if statements or while statements to deal with error checking to avoid an exception.
- `traceback` is a lengthy error message that will give information with respect to the line number that caused the exception.
- last line of the error message gives the name of the exception that was raised (i.e `ZeroDivisionError, ValueError`)
- `try` suite are statements that can potentially raise an exception
- `except` clause contains a block of statements that is called the `handler`

Exception syntax

```python
try:
  statement
  statement
except ExceptionName:
  statement
  statement
```

Example

```python
def main():
  try:
    hours = int(input('How many hours did you work?'))
    pay_rate = float(input('Enter your hourly pay rate: '))
    gross_pay = hours * pay_rate

    print('Gross pay: $ %.2f' % gross_pay, sep='')

  except ValueError:
    print('ERROR: hours worked and hourly pay rate must be valid
        integers.')


main()
```

Lists and Tuples

- A list is mutable which means a program can change its contents.
- A tuple is immutable which means that once it is created its contents cannot be changed.

Introduction to Lists

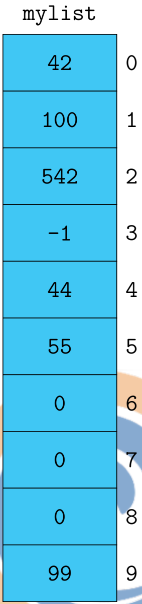
- a list is an object that contains multiple data items
- their contents can be changed during runtime
- lists are dynamic meaning you can add or remove items from them
- indexing, slicing, and other methods will be introduced
- a *list* is an object that contains multiple data items
- a member of a *list* is called an *element*
- Can access elements directly or via index

```python
even_numbers = [2, 4, 6, 8, 10]
names = ['Molly', 'Steven', 'Will', 'Alicia', 'Adriana']
info = ['Alicia', 27, 1550.87]
print(info)
numbers = list(range(5))
numbers2 = list(range(1,10,2))
```

Lists

- List indexing Start counting from 0
  - First element has index 0
  - Last element has index *Size - 1*
- Example

```
mylist[0] = 42
mylist[1] = 100
mylist[2] = mylist[0] + 5*mylist[1]
mylist[3] = -1
mylist[4] = 44
mylist[5] = 55
// ...
mylist[9] = 99
```



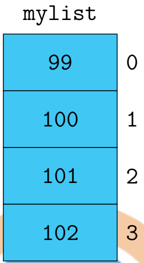
Iterating over a list with a `for` loop

- Accessing elements of a list
  - `for` loops are very helpful

    ```
    for ii in mylist:
       ...ii...
    ```
- traverses list elements directly
- Example

```
def main():
  mylist = [99, 100, 101, 102]
  for ii in mylist:
    print(ii)
 main()
```

```
99
100
101
102
```



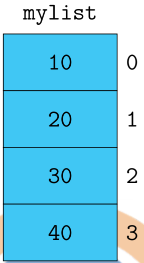
Iterating over a list with a `while` loop

- Accessing elements of a list
  - counter-controlled `while` loops are very helpful

    `while ii < length`

    `...mylist[ii]...`
  - traverses list elements by index.

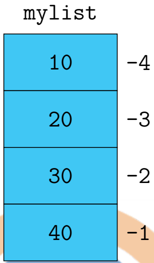
```
def main():
  mylist = [10, 20, 30, 40]
  size = len(mylist)
  ii = 0
  while ii < size:
    print(mylist[ii])
    ii += 1
main()
```



```
10
20
30
40
```

Iterating over a list with a `while` loop

- Accessing elements of a list
  - counter-controlled `while` loops are very helpful
    `while ii < length`
    `...mylist[ii]...`
  - traverses list elements by index.



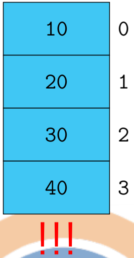
```python
def main():
  mylist = [10, 20, 30, 40]
  size = len(mylist)
  ii = -1
  while ii >= -size:
    print(mylist[ii])
    ii -= 1
main()
```

```
40
30
20
10
```

IndexError

- list indexing
- An IndexError exception will be raised if you use an invalid index with a list.
- Example

```
def main():
  mylist = [10, 20, 30, 40]
  ii = 0
  while ii <= len(mylist):
    print(mylist[ii])
    ii += 1
main()
```



Lists are mutable

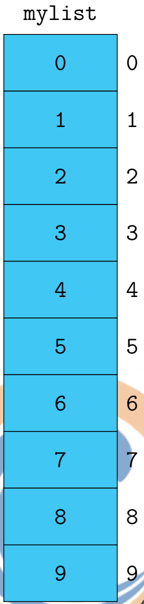
- Lists in Python are mutable, which means their elements can be changed.
- An expression in the form `list[index]` can appear on the left side of an assignment operator.

```
SIZE = 10
def main():
  mylist = list(range(0,SIZE))

  for ii in mylist:
    mylist[ii] = ii*10 + ii

 print(mylist)

main()
```



Lists are mutable

- Lists in Python are mutable, which means their elements can be changed.
- An expression in the form `list[index]` can appear on the left side of an assignment operator.

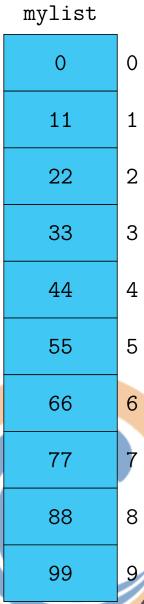
```
SIZE = 10
def main():
  mylist = list(range(0,SIZE))

  for ii in mylist:
    mylist[ii] = ii*10 + ii

 print(mylist)

main()
```

```
[0, 11, 22, 33, 44, 55, 66, 77, 88, 99]
```

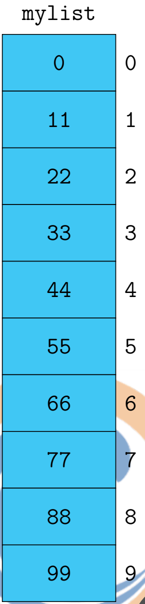


Filling a list with values

- To use indexing expressions to fill the list you must create a list first.
- For example, `mylist = [0]*5` creates a list with 5 elements where each element is assigned the value 0.

```
SIZE = 10
def main():
  mylist = [0]*SIZE
  ii = 0
  while ii < SIZE:
    mylist[ii] = ii*10 + ii
    ii += 1
  print(mylist)
main()
```

```
[0, 11, 22, 33, 44, 55, 66, 77, 88, 99]
```



User input assigned to members of a list

sales_list.py

```python
NUM_DAYS = 5
def main():
  sales = [0]*NUM_DAYS
  index = 0
  print('Enter the sales for each day.')
  while index < NUM_DAYS
    print('Day # %d : ' % (index+1), sep = '', end='')
    sales[index] = float(input())
    index+=1

  print('Here are the values you entered: ')
  for value in sales:
    print('%.2f' % value)

main()
```

```
1000.00
2000.00
3000.00
4000.00
5000.00
```

Unpacking a List

```python
x = ['red', 'yellow', 'blue']
a,b,c = x
```

- The number of variables must match the length of the list.
- `a='red'`
- `b='yellow'`
- `c='blue'`

enumerate():

```python
x = ['red', 'yellow', 'blue']
for ii, item in enumerate(x):
  print(ii,item)
```

```
0 red
1 yellow
2 blue
```

Constructing a list

```
>>> x = list()
>>> x = ['hi mom', 3, 4.7, 'whoa nelly!']
>>> x = list(tuple1)
```

List Comprehension

```
>>> x = [m for m in range(8)]
>>> x
[0, 1, 2, 3, 4, 5, 6, 7]
>>> x = [z**2 for z in range(10) if z>4]
>>> x
[25, 36, 49, 64, 81]
```

Concatenating Lists

You can concatenate lists *only* with other lists

```python
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
list1 += list2
print(list1)
print(list2)

girl_names = ['Joanne', 'Karen', 'Lori']
girl_names += ['Jenny', 'Kelly']
print(girl_names)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
[5, 6, 7, 8]
['Joanne', 'Karen', 'Lori', 'Jenny', 'Kelly']
```

List Slicing

- A slicing expression selects a range of elements from a sequence/list
- Syntax `list_name[start:end]`
- `start` is the starting index of the first element in the slice
- `end` is the index marking the end of the slice that `start` up to but do NOT include the `end`.
- If `start`>`end` the slicing expression will return an empty list

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', '
    Friday', 'Saturday']
mid_days = days[2:5]
print(mid_days)
```

```
['Tuesday', 'Wednesday', 'Thursday']
```

Finding Items in Lists with `in`

```python
def main():
  prod_nums = ['V475', 'F987', 'Q143', 'R688']
  search = input('Enter a product number: )
  if search in prod_nums:
    print('%s was found in the list' % search)
  else:
    print('%s was not found in the list' % search)

main()
```

```python
def main():
  prod_nums = ['V475', 'F987', 'Q143', 'R688']
  search = input('Enter a product number: )
  if search not in prod_nums:
    print('%s was not found in the list' % search)
  else:
    print('%s was found in the list' % search)

main()
```

Method	Description
`append(item)`	Adds *item* to the end of the list.
`index(item)`	Returns the index of the first element whose value is equal to item. A `ValueError` exception is raised if item is not found in the list.
`insert(index, item)`	Inserts *item* into the list at the specified *index* . When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list. No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
`sort()`	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
`remove(item)`	Removes the first occurrence of *item* from the list. A *ValueError* exception is raised if item is not found in the list.
`reverse()`	Reverses the order of the items in the list.

list_append.py

```python
def main():
  # First, create an empty list.
  name_list = []
  again = 'y'
  while again == 'y':
     name = input('Enter a name: ')
     name_list.append(name)
     print('Do you want to add another name?')
     again = input('y = yes, anything else = no: ')
     print()
  print('Here are the names you entered.')
  for name in name_list:
    print(name)

main()
```

Copying Lists

- To make a copy of a list, you must copy the list's elements
- Assignment will create two variables that will reference the same list in memory

```python
list1 = [1, 2, 3, 4]
list2 = list1 # creates two variables that reference the same
    list
```

- Must copy the list and then reference two separate but identical lists

```python
list1 = [1, 2, 3, 4]
list2 = []
for item in list1:
  list2.append(item)

#alternatively
list3 = [] + list1
```

Processing Lists

```python

NUM_EMPLOYEES = 6

def main():
  hours = [0] * NUM_EMPLOYEES
  for index in range(NUM_EMPLOYEES):
    print('Enter the hours worked by employee ', index + 1, ': ',
        sep='', end='')
    hours[index] = float(input())

  pay_rate = float(input('Enter the hourly pay rate: '))

  for index in range(NUM_EMPLOYEES):
    gross_pay = hours[index] * pay_rate
    print('Gross pay for employee ', index + 1, ': $', format(
        gross_pay, ',.2f'), sep='')

main()
```

Processing Lists `total_list.py`

```python
def main():
  scores = [2.5, 7.3, 6.5, 4.0, 5.2]
  total = 0.0
  # Calculate the total of the list elements.
  for value in scores:
    total += value
  # Calculate the average of the elements.
  average = total / len(scores)
  # Display the total of the list elements.
  print('The average of the elements is', average)
  # Call the main function.
main()
```

Processing Lists `total_function.py`

```python
def main():
  # Create a list.
  numbers = [2, 4, 6, 8, 10]
  # Display the total of the list elements.
  print('The total is', get_total(numbers))

def get_total(value_list):
  # Create a variable to use as an accumulator.
  total = 0
  # Calculate the total of the list elements.
  for num in value_list:
    total += num
 # Return the total.
  return total
# Call the main function.
main()
```

Returning a List from Function `return_list.py`

```python
def main():
  numbers = get_values()
  print('The numbers in the list are:')
  print(numbers)

def get_values():
  # Create an empty list.
  values = []
  # Create a variable to control the loop.
  again = 'y'
  while again == 'y':
    num = int(input('Enter a number: '))
    values.append(num)
    # Want to do this again?
    print('Do you want to add another number?')
    again = input('y = yes, anything else = no: ')
    print()
  # Return the list.
  return values
# Call the main function.

main()
```

Processing Lists Problem

Design a program that will read student test scores as input and calculates the average with the lowest score dropped. To do this you must:

- Get the student's test scores.
- Calculate the total of the scores.
- Find the lowest score.
- Subtract the lowest score from the total to get the adjusted total.
- Divide adjusted total by one less than the number of test scores.
- Display the average.

Processing Lists `drop_lowest_score.py` p1/2

```
def main():
  scores = get_scores()
  total = get_total(scores)
  lowest = min(scores)
  # Subtract the lowest score from the total.
  total -= lowest
  average = total / (len(scores) - 1)
  print('The average, with the lowest score dropped', 'is:',
      average)

def get_scores():
  test_scores = []
  again = 'y'
  while again == 'y':
    value = float(input('Enter a test score: '))
    test_scores.append(value)
    print('Do you want to add another score?')
    again = input('y = yes, anything else = no: ')
    print()
  return test_scores
```

Processing Lists `drop_lowest_score.py` p2/2

```
# The get_total function accepts a list as an
# argument returns the total of the values in
# the list.
def get_total(value_list):
  total = 0.0
  for num in value_list:
    total += num
  return total

# Call the main function.
main()
```

Writing a list to a file

```python
def main():
  cities = ['New York', 'Boston', 'Atlanta', 'Dallas']
  # Open a file for writing.
  outfile = open('cities.txt', 'w')
  # Write the list to the file.
  for item in cities:
    outfile.write(item + '\n')
  # Close the file.
  outfile.close()
# Call the main function.
main()
```

```
New York
Boston
Atlanta
Dallas
```

Reading a list from a file

```
def main():
  # Open a file for reading.
 infile = open('numberlist.txt', 'r')

  numbers = infile.readlines()
  infile.close()

  index = 0
  while index < len(numbers):
    numbers[index] = int(numbers[index])
    index += 1

  print(numbers)
main()
```

Nested/2D Lists

```
>>> students = [['Joe', 'Kim'],['Sam', 'Sue'],['Kelly', 'Chris']]
>>> print(students)
[['Joe', 'Kim'], ['Sam', 'Sue'], ['Kelly', 'Chris']]
>> print(students[0])
['Joe', 'Kim']
```

'Joe'	'Kim'
'Sam'	'Sue'
'Kelly'	'Chris'

Nested/2D Lists

```python

# This program assigns random numbers to
# a two-dimensional list.
import random
# Constants for rows and columns
ROWS = 3
COLS = 4
def main():
  # Create a two-dimensional list.
  values = [[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0]]
  # Fill the list with random numbers.
  for r in range(ROWS):
    for c in range(COLS):
      values[r][c] = random.randint(1, 100)
  print(values)
# Call the main function.
main()
```

Tuples

- A tuple is an immutable sequence, which means that its contents cannot be changed.
- Like lists, tuples support indexing.
- Tuples support all the same operations as lists, except those that change the contents of the list.
- Tuples do not support methods such as append, remove, insert, reverse, and sort.
- If you want to create a tuple with just one element, you must write a trailing comma after the element's value `my_tuple = (1,)`

```
>>> number_tuple = (1,2,3)
>>> number_list = list(number_tuple)
>>> print(number_list)
[1,2,3]
>>> str_list = ['one', 'two', 'three']
>>> str_tuple = tuple(str_list)
>>> print(str_tuple)
('one', 'two', 'three')
```

Strings

- Can access each letter in a string directly
- Can access each letter in a string by index

```
name = 'Juliet'
for ch in name:
  print(ch)
```

```
J
u
l
i
e
t
```

Strings

- Strings are immutable
- Can access each letter in a string directly
- Can access each letter in a string by index

```
city = 'Boston'
index = 0
while index < len(city):
  print(city[index])
  index += 1
```

```
B
o
s
t
o
n
```

concatenate.py

```
def main():
  name = 'Carmen'
  print('The name is', name)
  name = name + ' Brown'
  print('Now the name is', name)

# Call the main function.
main()
```

```
The name is Carmen
Now the name is Carmen Brown
```

String Slicing

```
def get_login_name(first, last, idnumber):
  # Get the first three letters of the first name.
  # If the name is less than 3 characters, the
  # slice will return the entire first name.
  set1 = first[0 : 3]
  # Get the first three letters of the last name.
  # If the name is less than 3 characters, the
  # slice will return the entire last name.
  set2 = last[0 : 3]
  # Get the last three characters of the student ID.
  # If the ID number is less than 3 characters, the
  # slice will return the entire ID number.
  set3 = idnumber[-3 :]
  # Put the sets of characters together.
  login_name = set1 + set2 + set3
  # Return the login name.
  return login_name
```

Testing, Searching, and Manipulating Strings

- Testing Strings with `in` and `not in`

```
text = 'Four score and seven years ago'
if 'seven' in text:
  print('The string "seven" was found.')
else:
  print('The string "seven" was not found.')
```

- String methods are functions that belong to an object and performs operations on that object with the syntax `stringvar.method(arguments)`
  - Testing values of strings
  - Performing modifications on strings
  - Searching for substrings and replacing sequences of characters

StringMethods

Method	Description
`isalnum()`	Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.
`isalpha()`	Returns true if the string contains only alphabetic letters and is at least one character in length. Returns false otherwise.
`isdigit()`	Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.
`islower()`	Returns true if all of the alphabetic letters in the string are lowercase, and the string contains at least one alphabetic letter. Returns false otherwise.
`isspace()`	Returns true if the string contains only whitespace characters and is at least one character in length. Returns false otherwise. (Whitespace characters are spaces, newlines (`\n`), and tabs (`\t`).
`isupper()`	Returns true if all of the alphabetic letters in the string are uppercase, and the string contains at least one alphabetic letter. Returns false otherwise.

`lower()`	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.
`lstrip()`	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the beginning of the string.
`lstrip(char)`	The char argument is a string containing a character. Returns a copy of the string with all instances of char that appear at the beginning of the string removed.
`rstrip()`	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the end of the string.
`rstrip(char)`	The char argument is a string containing a character. The method returns a copy of the string with all instances of char that appear at the end of the string removed.
`strip()`	Returns a copy of the string with all leading and trailing whitespace characters removed.
`strip(char)`	Returns a copy of the string with all instances of char that appear at the beginning and the end of the string removed.
`upper()`	Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase, or is not an alphabetic letter, is unchanged.

`lower()`	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.
`lstrip()`	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the beginning of the string.
`lstrip(char)`	The char argument is a string containing a character. Returns a copy of the string with all instances of char that appear at the beginning of the string removed.
`rstrip()`	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the end of the string.
`rstrip(char)`	The char argument is a string containing a character. The method returns a copy of the string with all instances of char that appear at the end of the string removed.
`strip()`	Returns a copy of the string with all leading and trailing whitespace characters removed.
`strip(char)`	Returns a copy of the string with all instances of char that appear at the beginning and the end of the string removed.
`upper()`	Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase, or is not an alphabetic letter, is unchanged.

`endswith(substring)`	The substring argument is a string. The method returns true if the string ends with substring.
`find(substring)`	The substring argument is a string. The method returns the lowest index in the string where substring is found. If substring is not found, the method returns -1.
`replace(old,new)`	The old and new arguments are both strings. The method returns a copy of the string with all instances of old replaced by new.
`startswith(substring)`	The old and new arguments are both strings. The method returns a copy of the string with all instances of old replaced by new.