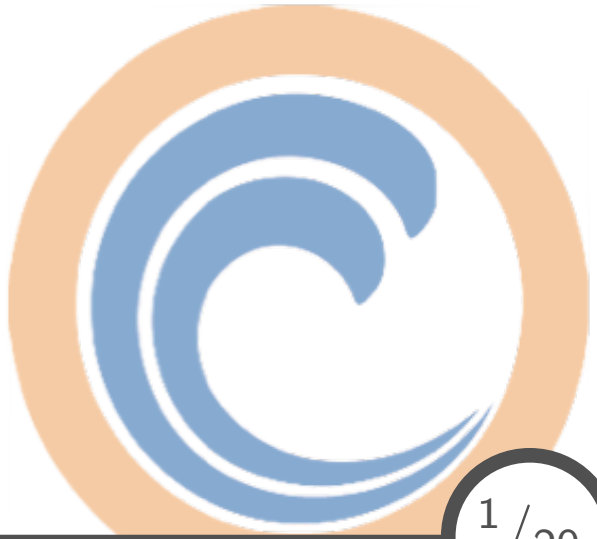
CS A131: Lecture 8

Nadia Ahmed

Orange Coast College, Computer Science

CS A131



Lecture 8: Overview

- Functions that Return Multiple Values
- File I/O
  - Introduction
  - Using loops to process files
  - Processing records
  - Exceptions (next lecture)

Functions That Return Multiple Values

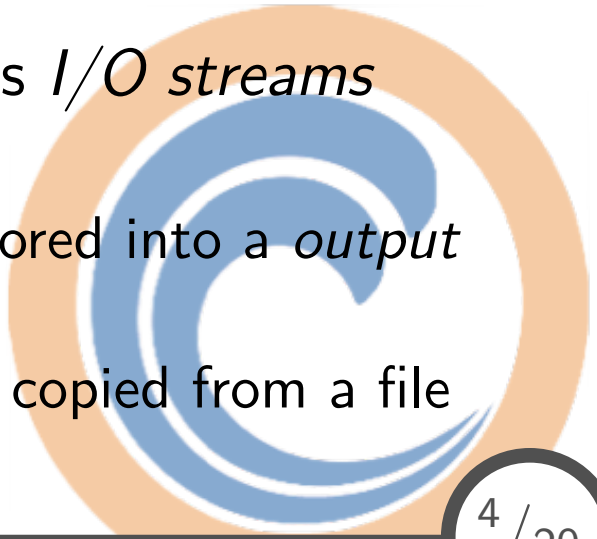
- Examples of value returning functions show only a single value returned after the `return` statement
- In Python you are not limited to returning only one value.
- Specify multiple expressions separated by commas after the `return` statement.
- The number of variables on the left side of the = operator from the calling function must match the number of values returned by the called function

```
1  def get_name():
2    first = input('Enter your first name: ')
3    last = input('Enter your last name: ')
4    return first, last
5
6  def main():
7    first_name, last_name = get_name()
8    print('Your name is %s %s' % (first_name, last_name))
9
10 main()
```

File Processing

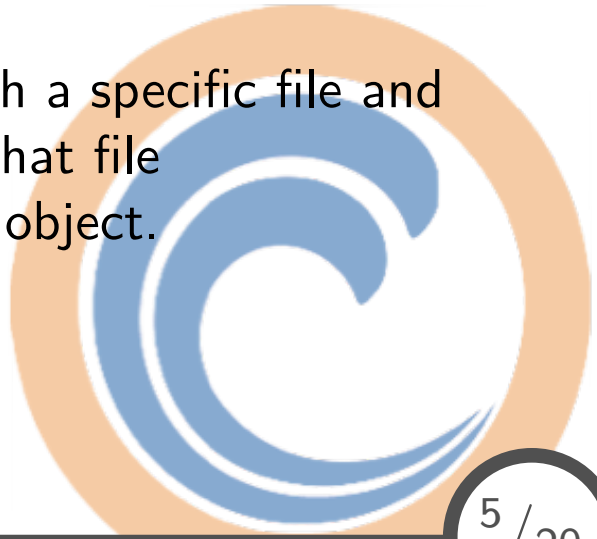
Introduction

- Up to now, all data processed is available only during program run time
  - At program completion, all data is lost because it is stored in RAM
- *Persistent data* is stored even after a program exits
- Persistent data is stored in files...
  - on the harddisk
  - on a removable disk (CD, memory stick, etc)
  - on a tape
- Input and output from/to files is organized as *I/O streams*
- Saving and retrieving data
  - *Write* data $\rightarrow$ variables stored in RAM are stored into a *output* file
  - *Read* data from an input file $\rightarrow$ variables are copied from a file into RAM and referenced by a variable.



Introduction to File Input and Output

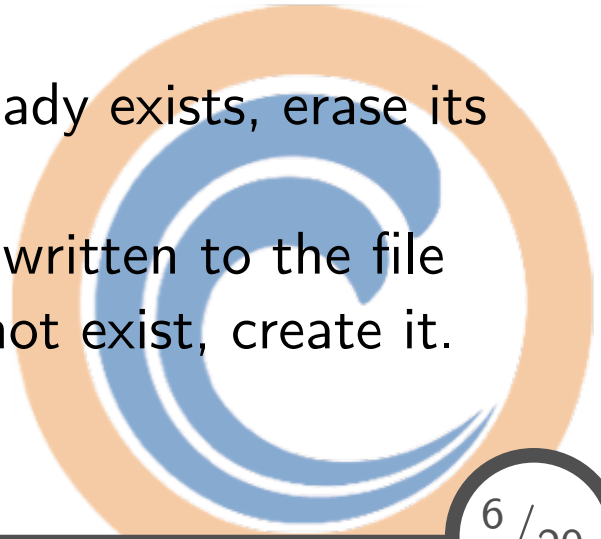
- Open the file
  - text files
  - binary files
- Process the file
  - Access
    - Sequential access (must read from beginning to end)
    - Direct access/random access file (can jump to a piece of data in a file)
  - File object
    - a *file object* is an object that is associated with a specific file and provides a way for the program to work with that file
    - In the program the variable references the file object.
- Close the file



Opening a file

```
1 file_variable = open(filename, mode)
```

- `file_variable` is the name of the variable that will reference the file object
- `filename` is the string specifying the name of the file
- `mode` is a string specifying the mode in which the file will be opened
  - `'r'` → Open a file for reading only. The file cannot be changed or written to.
  - `'w'` → Open a file for writing. If the file already exists, erase its contents. If it does not exist create it.
  - `'a'` → Open a file to be written to All data written to the file will be appended to its end. If the file does not exist, create it.



Specifying the Location of a File

```
1 #file that is in current directory
2 test_file = open('test.txt', 'w')
3 #file that is in a different directory
4 test_file2 = open(r'C:\Users\Blake\temp\test2.txt', 'w')
```

r specifies that the string is a raw string which allows the interpreter to read the backslash characters as literal backslashes instead of escape sequences.

Writing Data to a File

```
1 file_variable.write(string)
```

- `file_variable` is a variable that references a file object.
- `string` is a string that will be written to the file.
- The file must be opened for writing using `'w'` or `'a'` otherwise an error will occur.
- If using `'w'` you will overwrite the contents of the file if there was any.
- Close the file to avoid data loss.

```
1 test_file.write('Hi Mom!')
2
3 #alternatively
4 my_string = 'Hi Dad!'
5 test_file2.write(my_string)
6 test_file.close()
```

Writing Data to a File `file_write.py`

```
1  def main():
2    outfile = open('philosophers.txt', 'w')
3
4    outfile.write('John Locke\n')
5    outfile.write('David Hume\n')
6    outfile.write('Edmund Burke\n')
7
8    outfile.close()
9
10 main()
```

readline() example line_read.py

```
1  def main():
2    infile = open('philosophers.txt', 'r')
3
4    line1 = infile.readline()
5    line2 = infile.readline()
6    line3 = infile.readline()
7
8    infile.close()
9
10   print(line1)
11   print(line2)
12   print(line3)
13
14 main()
```

```
1 John Locke
2
3 David Hume
4
5 Edmund Burke
```

Output to screen shows 2 carriage returns between each line printed. Why?

Concatenating a Newline to String

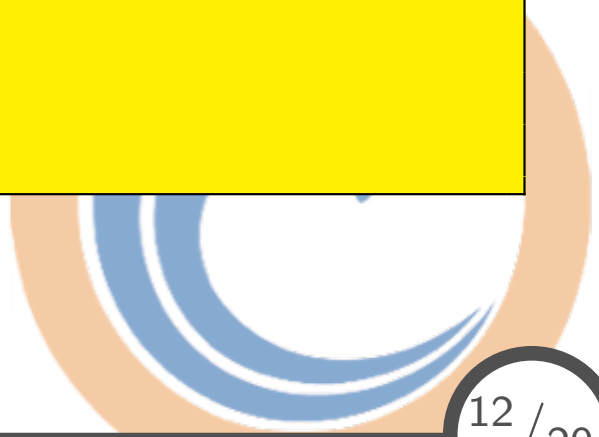
write_names.py

```
1  def main():
2    print('Enter the names of three friends.')
3    name1 = input('Friend 1: ')
4    name2 = input('Friend 2: ')
5    name3 = input('Friend 3: ')
6
7    myfile = open('friends.txt', 'w')
8
9    myfile.write(name1 + '\n')
10   myfile.write(name2 + '\n')
11   myfile.write(name3 + '\n')
12
13   myfile.close()
14   print('The names were written to friends.txt')
15 main()
```

```
1  Enter the names of three friends.
2  Friend 1: Ringo
3  Friend 2: Paul
4  Friend 3: John
5  The names were written to friends.txt
```

rstrip example strip_newline.py

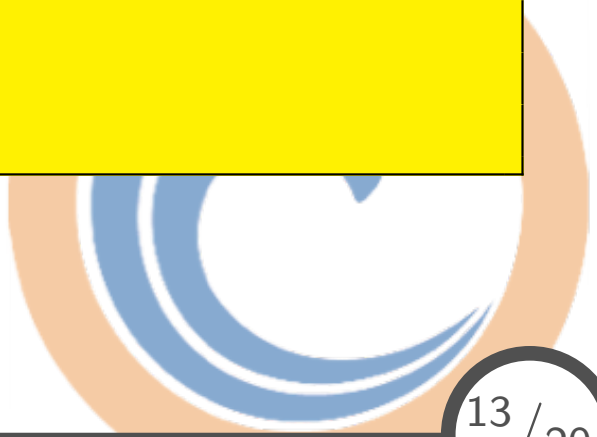
```
1  def main():
2    infile = open('philosophers.txt', 'r')
3
4    line1 = infile.readline()
5    line2 = infile.readline()
6    line3 = infile.readline()
7
8    line1 = line1.rstrip('\n')
9    line2 = line2.rstrip('\n')
10   line3 = line3.rstrip('\n')
11
12   infile.close()
13   print(line1)
14   print(line2)
15   print(line3)
16
17 main()
```



Reading numbers `read_numbers.py`

Convert string from file to integer or floating point value to use for processing

```
1  def main():
2    infile = open('numbers.txt', 'r')
3
4    num1 = int(infile.readline())
5    num2 = int(infile.readline())
6    num3 = int(infile.readline())
7
8    infile.close()
9
10   total = num1 + num2 + num3
11
12   print('The numbers are: %d %d %d' % (num1, num2, num3))
13   print('Their total is: %d' % total)
14
15 main()
```

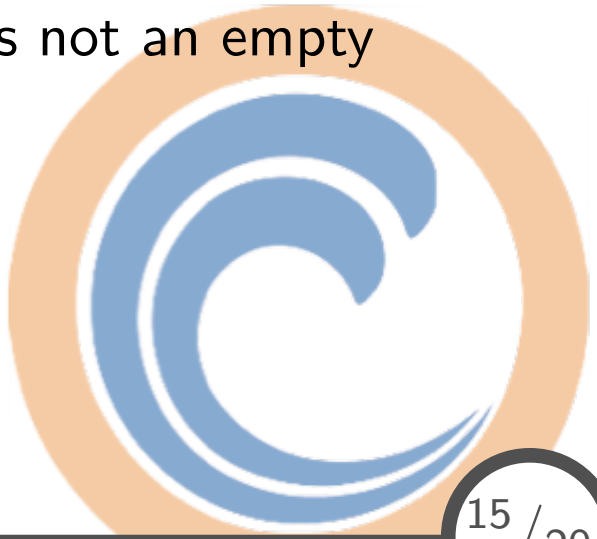


Using Loops to Write to a File write_sales.py

```
1  def main():
2    num_days = int(input('For how many days do you have sales?'))
3    sales_file = open('sales.txt', 'w')
4
5    for count in range(1, num_days+1):
6      sales = float(input('Enter the sales for day #%d :' % count))
7      sales_file.write(str(sales) + '\n'))
8
9    sales_file.close()
10   print('Data written to sales.txt')
11
12 main()
```

Using Loops to Read from a File `read_sales.py`

- `readline()` allows you to detect the end of a long file.
- `readline` returns an empty string ‘ ’ when it has read beyond the end of the file
- We can use a `while` loop to determine the end of a file
- Procedure:
  - Open the file
  - Use `readline()` to read the first line
  - While the value returned from `readline()` is not an empty string
    - Process the item read
    - `readline()` to read next line from file



Using Loops to read from a File `read_sales.py`

`while` loop

```
1  def main():
2    sales_file = open('sales.txt', 'r')
3
4    # read first line of file w/o converting to a number
5    # b/c it could be an empty string
6    line = sales_file.readline()
7
8    while line != '':
9      amount = float(line)
10     print('amount: %.2f' % amount)
11
12     #read next line
13     line = sales_file.readline()
14
15   sale_file.close()
16
17 main()
```

Using Loops to read from a File `read_sales2.py`

`for` loop

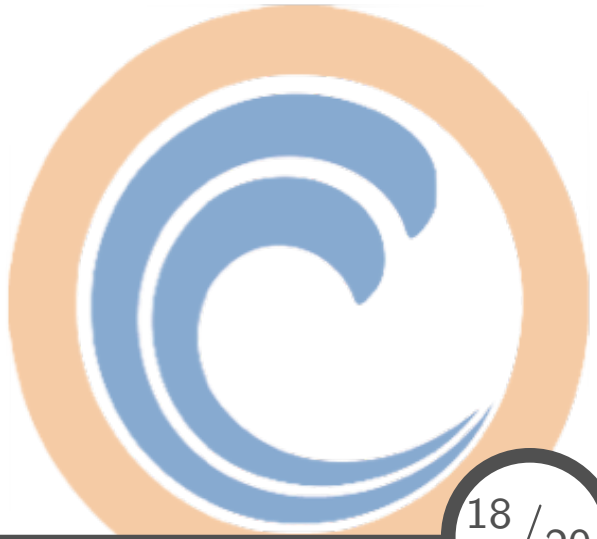
```
1  # syntax
2  for variable in file_object:
3    statement
4    statement
```

- `file_object` is the name of the variable that references the file object
- loop iterates once for each line in file

```
1  def main():
2    sales_file = open('sales.txt', 'r')
3
4    for line in sales_file:
5      # convert line into a float
6      amount = float(line)
7      # format and display amount
8      print('amount: %.2f' % amount)
9
10   # close the file
11   sale_file.close()
12 main()
```

Processing Records

- A *record* is a complete set of data that describes one item
- A *field* is a single piece of data within a record
- Example: employee records may include
  - Name
  - ID
  - Department



Writing Records save_emp_records.py

```
1  def main():
2    num_emps = int(input('The number of employee records?'))
3    emp_file = open('employees.txt', 'w')
4    for count in range (1, num_emps+1):
5
6      print("Enter employee #', count, sep=' ')
7      name = input('name: ')
8      id_num = input('ID: ')
9      dept = input('dept: ')
10
11     emp_file.write(name + '\n')
12     emp_file.write(id_num + '\n')
13     emp_file.write(dept + '\n')
14
15     # print a blank line
16     print()
17
18   #close the file
19   emp_file.close()
20   print('Employee records written to employees.txt')
21
22 main()
```

Reading Records read_emp_records.py

```
1  def main():
2    emp_file = open('employees.txt', 'r')
3    name = emp_file.readline()
4
5    while name != '':
6      id_num = emp_file.readline()
7      dept = emp_file.readline()
8
9      #strip newlines from the fields
10     name = name.rstrip('\n')
11     id_num = id_num.rstrip('\n')
12     dept = dept.rstrip('\n')
13
14     #display records
15     print('Name: %s' %name)
16     print('ID: %s' % id_num)
17     print('Dept: %s' % dept)
18     print()
19     # update LCV
20     name = emp_file.readline()
21   #close file
22   emp_file.close()
23
24 main()
```