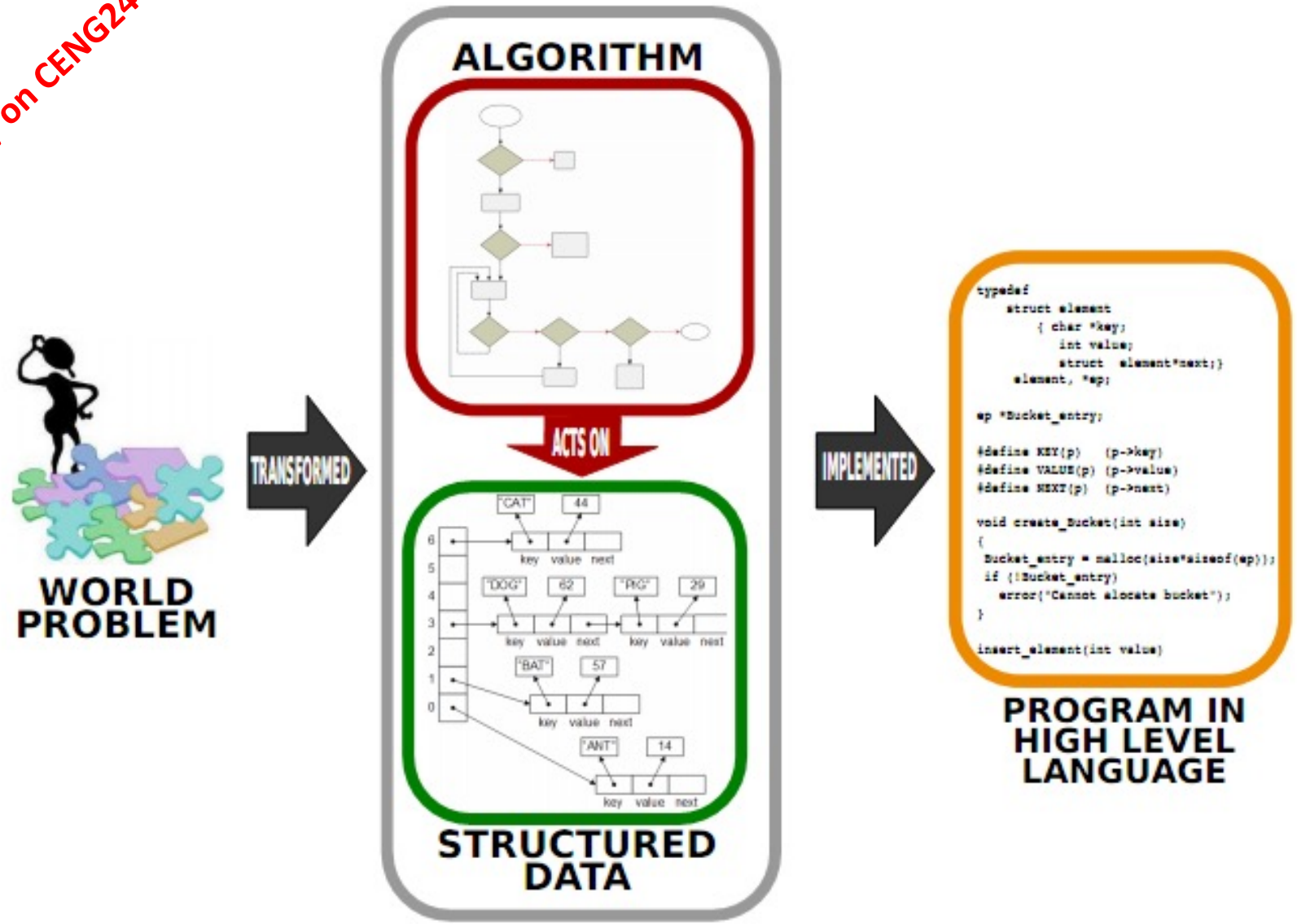
CEng 240 – Spring 2021
Week 11

Sinan Kalkan

File Handling

Disclaimer: Figures without reference are from either from "Introduction to programming concepts with case studies in Python" or "Programming with Python for Engineers", which are both co-authored by me.

Previously on CENG240!



What does OOP provide?

- Encapsulation: Hiding implementation/representation details

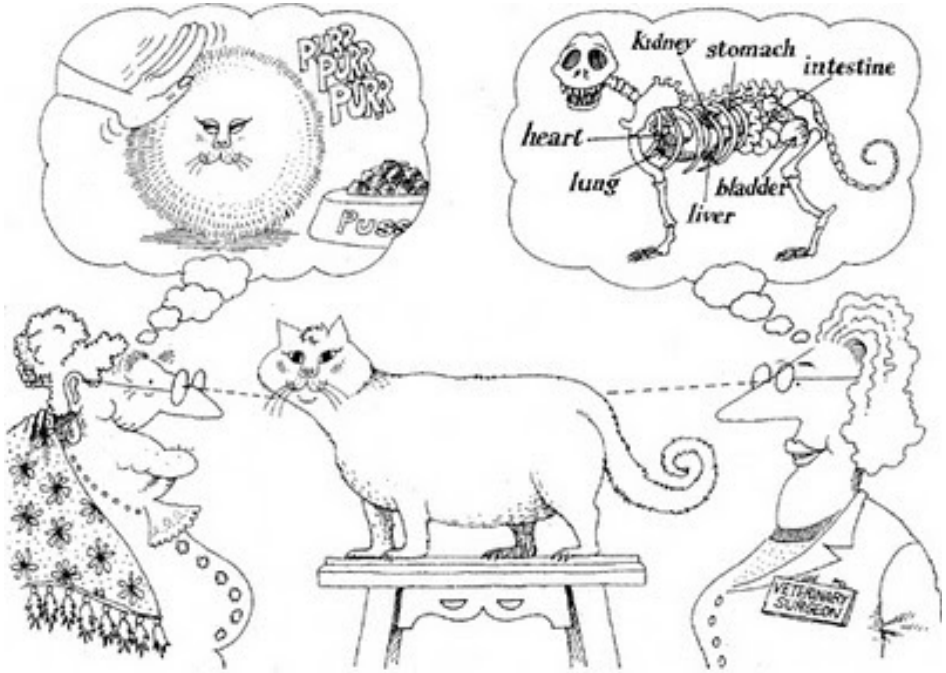
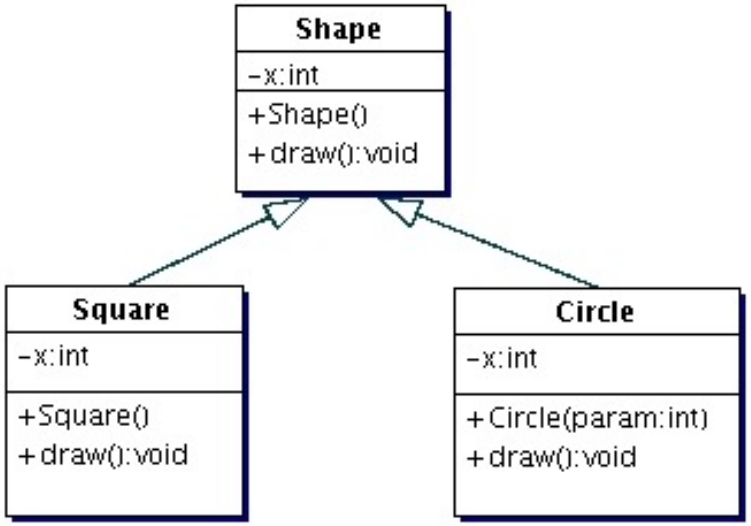


Figure: G. Booch, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen, K. A. Houston, Object-Oriented Analysis and Design with Applications (3rd Edition), 2007.

Previously on CENG240!

What does OOP provide?

- Inheritance:
  - A class inherits some variables and functions from another one.
  - Square class inherits x and draw() from the Shape class.
  - Shape: Parent class
  - Square: Child class



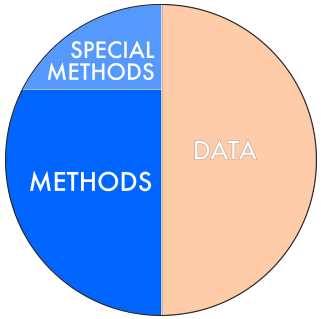
What does OOP provide?

- Polymorphism:
  - The ability of a child class to behave and appear like its parent.

```
class Animal:
    def __init__(self, name): #Constructor
        self.name = name
    def talk(self):
        pass # Overloaded by Child Classes

class Cat(Animal):
    def talk(self):
        return 'Meow'

class Dog(Animal):
    def talk(self):
        return 'Woof'

class Duck(Animal):
    def talk(self):
        return 'Quack'
```

Previously on CENG240!

Class Definition




```
class ClassName :
```

Statement block

```
class shape:
    color = None
    x = None
    y = None

    def set_color(self, red, green, blue):
        self.color = (red, green, blue)

    def move_to(self, x, y):
        self.x = x
        self.y = y
```

```
p = shape()
s = shape()
p.move_to(22, 55)
p.set_color(255, 0, 0)
s.move_to(49, 71)
s.set_color(0, 127, 0)
```

Inheritance in Python

```
class ClassName ( BaseClass1, BaseClass2, .. ):
    Statement block
```

This Week

- File Handling
  - Files and sequential access
  - Parsing
  - Termination of input
  - Formatting files
  - Binary files

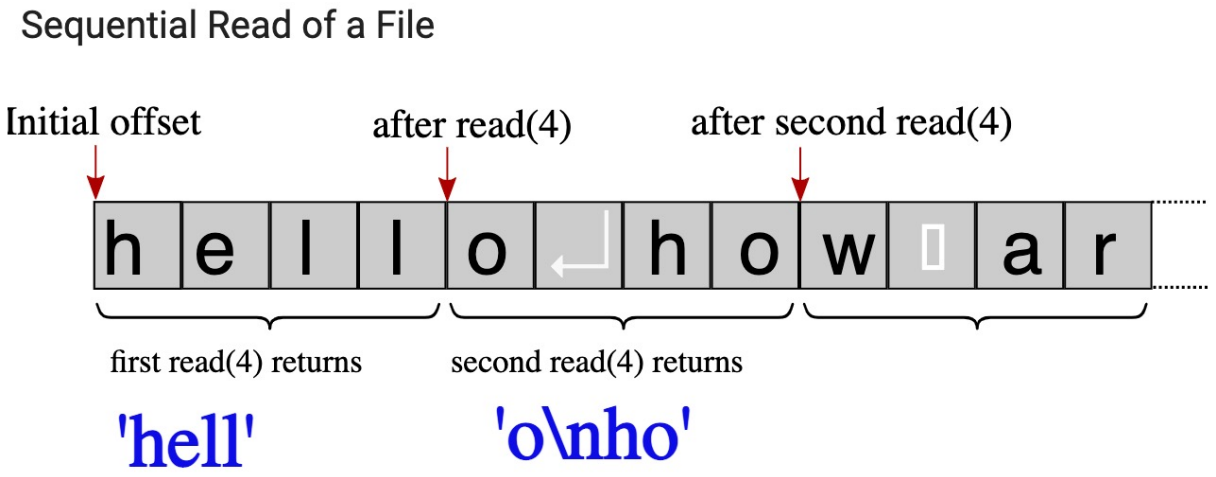
Administrative Notes

- Lab 7
- Midterm: 1 June, Tuesday, 17:40

First Example

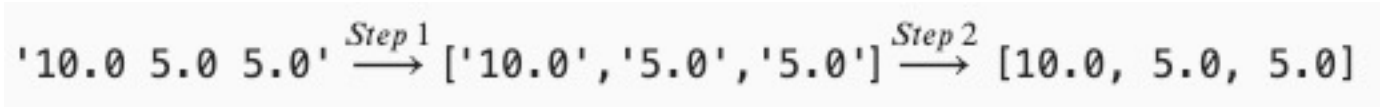
```
fpointer = open('firstexample.txt', "w")
fpointer.write("hello\n")
fpointer.write("how are\n")
fpointer.write("you?\n")
fpointer.close()
```

Files and Sequential Access




```
fp = open("firstexample.txt","r") # the example file we created above

for i in range(3): # repeat 3 times
        content = fp.read(4) # read 4 bytes in each step
        print("> ", content) # output 4 bytes preceded by >

fp.close()
```

Parsing




```
instr = '10.0 5.0 5.0’
outlst = []

# Go over each substring
for substr in instr.split(' ‘):
      outlst += [float(substr)]
      # Convert each element to float and append it to the list
```

OR: `outlst = [ float(substr) for substr in instr.split(' ')]`

Opening/closing files

- Opening files:
  - open(filename, “r”) => open file for reading
  - open(filename, “w”) => open file for writing
  - open(filename, “a”) => open file for appending

- Closing file:
  - fileobject.close()

Accessing Files Line by Line


```
pointlist = [(0,0), (10,0), (10,10), (0,10)]

fp = open("pointlist.txt", "w") # open file for writing
fp.write(str(len(pointlist))) # write list length
fp.write('\n’)

# Go over each point in the list
for (x,y) in pointlist: # for each x,y value in the list
        fp.write(str(x)) # write x
        fp.write(' ') # space as number separator
        fp.write(str(y)) # write y
        fp.write('\n') # \n as line separator

fp.close()
```

Produces file with content:

```
4
0 0
10 0
10 10
0 10
```

Accessing Files Line by Line

Read file with content:

```
4
0 0
10 0
10 10
0 10
```

```
fp = open("pointlist.txt") # open file for reading

nextline = fp.readline() # read the first line
while nextline != '': # while read is successful
        print(nextline) # output the line
        nextline = fp.readline() # read the nextline

fp.close() # when nextline == '' loop terminates
```

Termination of input

- There are two ways to stop reading input:
  1. By reading a definite number of items.
     - Call read() or readline() functions for a fixed number of times.
  2. By the end of the file.
     - Continue to read() or readline() until they return empty string ‘’.

```
fp = open("pointlist.txt") # open file for reading

nextline = fp.readline() # read the first line
while nextline != '': # until end of file
        ... # Do something  with the read line
        nextline = fp.readline() # read the next line
```

Termination of input

- We can also use our own special “marker”

File contents:

3 0
3.4 2.1
5.1 3.2
EOLIST
1 1.5
2.0 2.5

```
fp = open("twopointlists.txt")
pntlst1 = []                              # start with empty list
pntlst2 = []                              # start with empty list

nextline = fp.readline()                  # read the first line
while nextline != 'EOLIST\n':               # sentinel value
  nextline = nextline.rstrip('\n')        # remove occurrences of '\n' at the end
  (x, y) = nextline.split(' ')            # get x and y (note that they are still strings)
  x = float(x)                            # convert them into real values
  y = float(y)
  pntlst1.append( (x,y) )                 # add tuple at the end
  nextline = fp.readline()                # read the nextline

# first list has been read, now continue with the second list from the same file
nextline = fp.readline()
while nextline != '':                     # until end of file
  nextline = nextline.rstrip('\n')        # remove occurrences of '\n' at the end
  (x, y) = nextline.split(' ')            # get x and y (note that they are still strings)
  x = float(x)                            # convert them into real values
  y = float(y)
  pntlst2.append( (x,y) )                # add tuple at the end
  nextline = fp.readline()                # read the nextline

fp.close()
print('List 1:', pntlst1)
print('List 2:', pntlst2)
```

```
List 1: [(3.0, 0.0), (3.4, 2.1), (5.1, 3.2)]
List 2: [(1.0, 1.5), (2.0, 2.5)]
```

Formatting Files

- You can use the formatted strings to control the format of the strings you place in files:

<string>.format()

- For example:

```
'{:10}, {:20}, {:3d}, {:f7.3}'.format('Han', 'Solo', 80, -0.2)'
```

Binary Files

- A text file is human readable.
  - Each number is represented by the characters of the digits.
  - “3.14159265358979323846264338327950288” occupies 34 characters/bytes.
- Content of a binary file is directly the binary representation of data
  - Number 3.14159265358979323846264338327950288 occupies 4 bytes on a 32-bit computer.
  - But the file is not human-readable.
- Opening binary files: open(filename, ‘rb’) or open(filename, ‘wb’)
- struct module should be used for creating and interpreting bytes

Examples

- Read a CSV file

- Dice game exercise:

https://pp4e-workbook.github.io/chapters/file_handling/dice_game.html

Final Words: Important Concepts

- Sequential access. File access.
- Text files. Reading and writing text files. Parsing a text file.
- End of file, new line.
- Formatting files.
- Binary files and binary file access.

THAT'S ALL FOLKS!
STAY HEALTHY