CEng 240 – Spring 2021
Week 10

Sinan Kalkan

Object-Oriented Programming

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!

Higher-order functions

- map(function, Iterator)

```
>>> abs_it = map(abs, [10, -4, 20, -100])
>>> for x in abs_it: print(x)
10
4
20
100
```

- filter(predicate, Iterator)

```
>>> def positive(x): return x > 0
>>> for x in filter(positive, [10, -4, 20, -100]): print(x)
10
20
```

Recursion: an example (cont'd)

- Let us look at the pseudo-code:

$$N! = (N-1)! \times N \qquad N \in \mathbf{N},\ N > 0$$
$$0! = 1$$

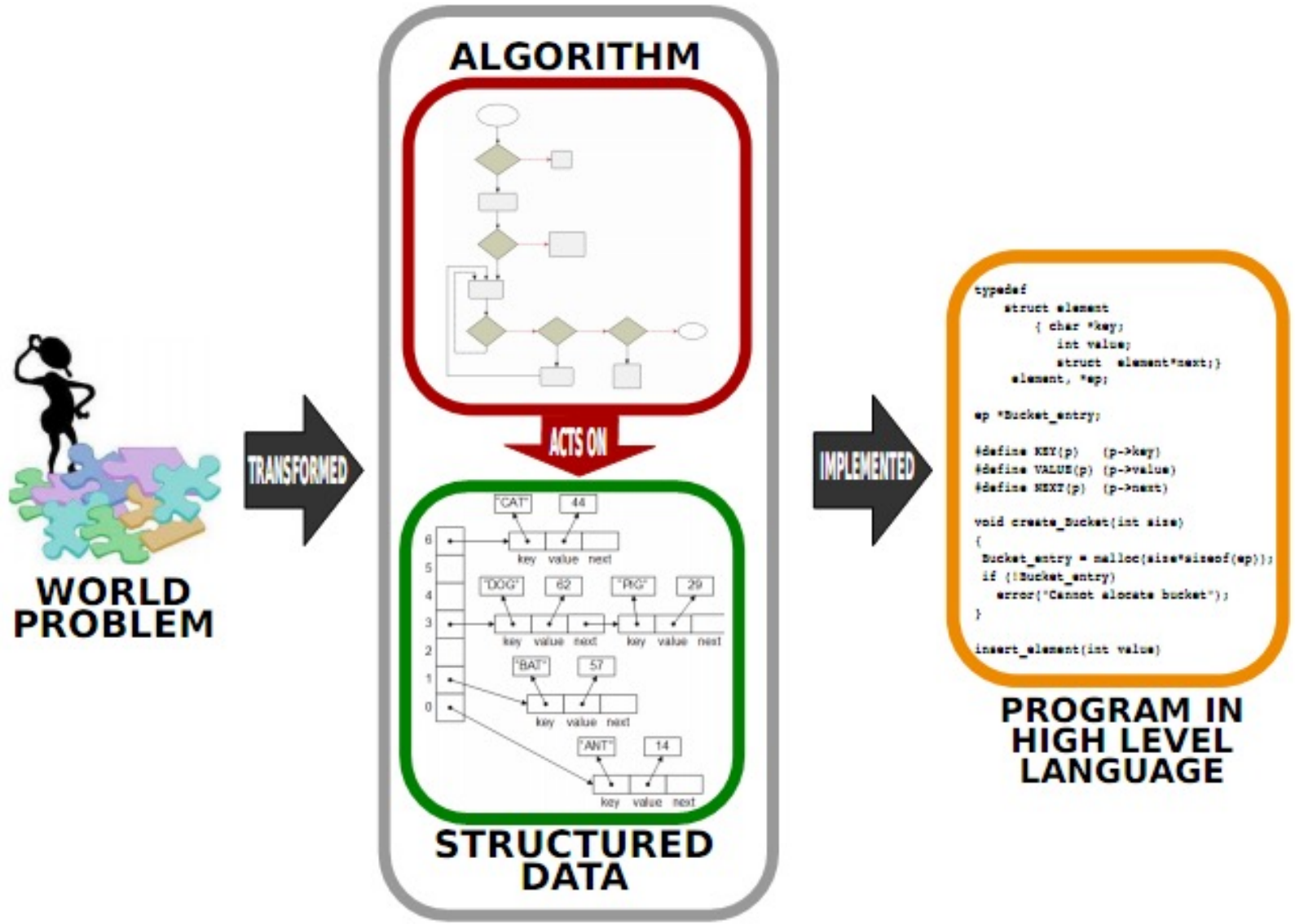
```
def factorial(N):
    if N == 0: return 1
    else: return N * factorial(N-1)
```

This Week

- Object-oriented Programming (OOP)
  - What is it? What are the benefits?
  - Properties of OOP: Encapsulation, Inheritance, Polymorphism
  - Class definition
  - Member functions and variables
  - The concept of message passing
  - Basics of OOP in Python

Administrative Notes

- Lab 6
- Midterm: 1 June, Tuesday, 17:40


WORLD PROBLEM
TRANSFORMED
ALGORITHM
ACTS ON
"CAT" 44
key value next
"DOG" 62
"PIG" 29
"BAT" 57
"ANT" 14
STRUCTURED DATA
IMPLEMENTED
typedef
struct element
{ char *key;
int value;
struct element*next;}
element, *ep;
ep *Bucket_entry;
#define KEY(p) (p->key)
#define VALUE(p) (p->value)
#define NEXT(p) (p->next)
void create_Bucket(int size)
{
Bucket_entry = malloc(size*sizeof(ep));
if (!Bucket_entry)
error("Cannot allocate bucket");
}
insert_element(int value)
PROGRAM IN HIGH LEVEL LANGUAGE

What is an object?

- An object is an entity which has a state and a set of behaviors that, when executed, change the state of the entity or the environment.

- A car object:
  - State:
    - Position, Speed, Gear State, Brake State, Wheel State
  - Behaviors:
    - Rotate, Accelerate, Brake

Why do we need/have OOP?

- Consider a problem of drawing/manipulating geometric objects:
  - Points in 2D Cartesian space
  - Lines: Made from two points
  - Triangle: Made from three lines or three points
  - Square/rectangle: Made from four lines or two points
  - Circle: A point and a radius
  - Polygon: A collection of lines / points.

Why do we need/have OOP? (cont'd)

- When we have to draw in 3D:
  - Points in 3D Cartesian space
  - 3D Lines: Made from two 3D points
  - 3D Triangle: Made from three 3D lines or three 3D points
  - 3D Square/3D rectangle: Made from four 3D lines or two 3D points
  - 3D Circle: A 3D point and a radius
  - Prism: A collection of 3D triangles & rectangles & parallelograms.

What does OOP provide?

- Encapsulation: Hiding implementation/representation details

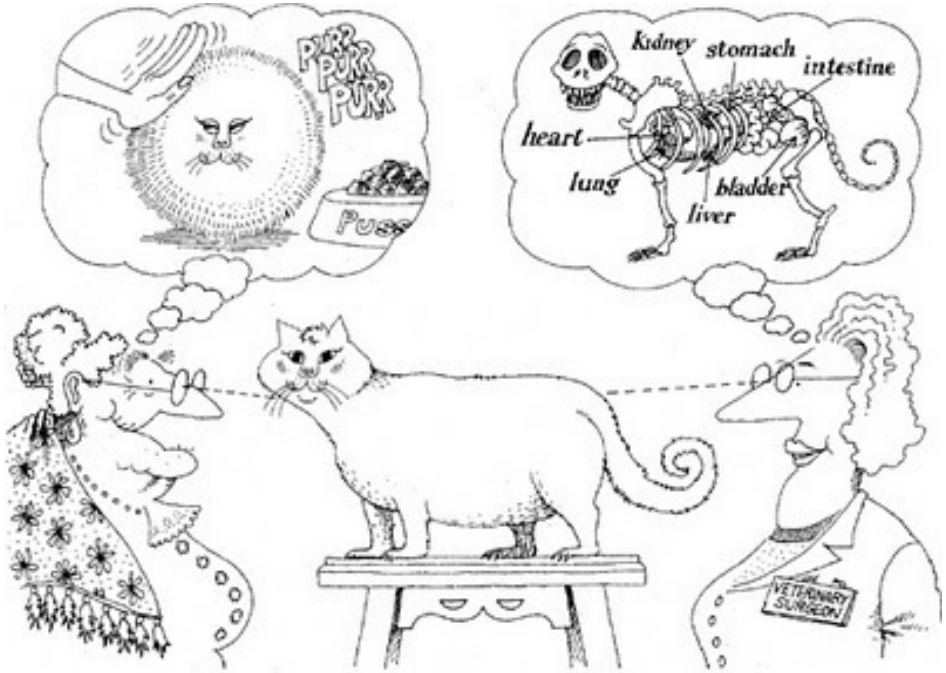
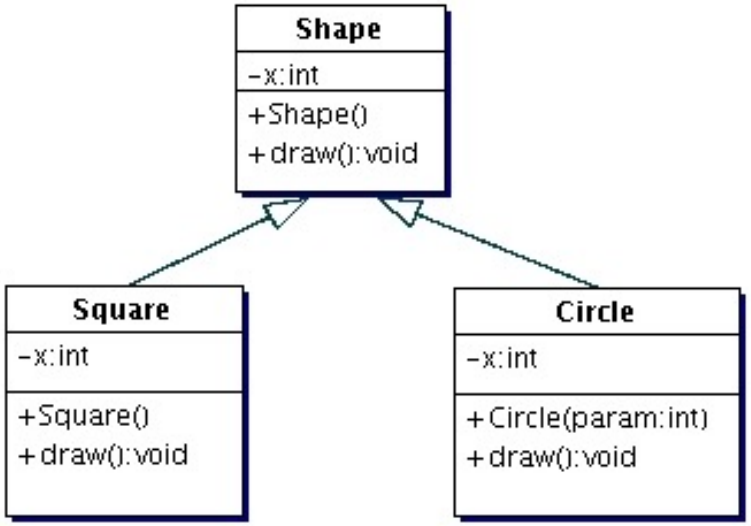


Figure: G. Booch, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen, K. A. Houston, Object-Oriented Analysis and Design with Applications (3rd Edition), 2007.

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.

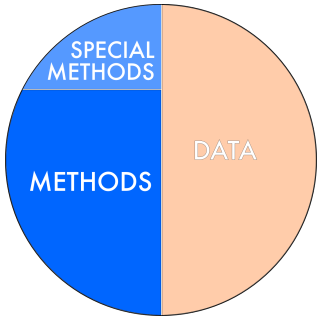
```
class Animal:
    def __init__(self, name): #Constructor
        self.name = name
    def talk(self):
        pass # Overloaded by Child Classes

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

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

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

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)
```

```
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

  def __str__(self):
    return "shape object: color=%s coordinates=%s" % (self.color, (self.x,self.y))

  def __init__(self, x, y):
    self.x = x
    self.y = y

  def __lt__(self, other):
    return self.x + self.y < other.x + other.y

p = shape(22,55)
s = shape(12,124)
p.set_color(255,0,0)
s.set_color(0,127,0)

print(s)
s.move_to(49,71)
print(s)

print(p.__lt__(s))
print(p < s)  # just the same as above but now infix

print(s.__dir__())
```

Special Methods

```
x<y   calls  x.__lt__(y)
x<=y  calls  x.__le__(y)
x==y  calls  x.__eq__(y)
x!=y  calls  x.__ne__(y)
x>y   calls  x.__gt__(y)
x>=y  calls  x.__ge__(y)
```

```
def __lt__(self, other):
    return self.x + self.y < other.x + other.y
```

Counter Example


```
class Counter:
  def __init__(self):
    self.value = 0    # this is the initialization

  def increment(self):
    '''increment the inner counter'''
    self.value += 1

  def get(self):
    '''return the counter as value'''
    return self.value

  def __str__(self):
    '''define how your counter is displayed'''
    return 'Counter:{}'.format(self.value)

stcnt = Counter()    # create the counter
stcnt.increment()
stcnt.increment()
print("# of students", stcnt)

sheep = Counter()
while sheep.get() < 1000:
  sheep.increment()

print("# of sheep",sheep)
```

```
# of students Counter:2
# of sheep Counter:1000
```

Inheritance in Python


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

Some Remarks

- If a member variable and a member function has the same name, the member variable override the member function.
- In Python, you cannot enforce data hiding.
- The users of a class should be careful about using member variables since they can be deleted or altered easily.
- To arrange proper deletion of an object, you can define a "__del__(self)" function ➔ also called, the destructor.
- The word self can be replaced by any other name. However, since "self" is the convention and some code browsers rely on the keyword "self", it is ideal to use "self" all the time.

Examples

- Person, Student, Instructor
- From the workbook
  - Database Recovery
  - https://pp4e-workbook.github.io/chapters/a_gentle_introduction_to_object_oriented_programming/database_recovery.html

Final Words: Important Concepts

- Encapsulation, inheritance and polymorphism.
- Benefits of the Object-Oriented Paradigm.
- Concepts such as class, instance, object, member, method, message passing.
- Concepts such as base class, ancestor, descendant.

THAT'S ALL FOLKS!
STAY HEALTHY