Department of Computer Technology and Information Systems

CTIS264 – Computer Algorithms

Spring 2019 - 2020

Lab Guide 6 - Week 8

Instructor : Füsun YÜRÜTEN

Assistant : Leyla SEZER

OBJECTIVE:

- **Python Classes**
- **Object Instances**
- **Defining and Working with Methods**
- **OOP Inheritance**
- **Empirical Analysis of Insertion-Sort algorithm.**
- **Topological Sort**
- **Lumuto's Partitioning (Quick Sort)**

Defining a class is simple in Python:

- You start with the class keyword to indicate that you are creating a class, then you add the name of the class (starting with a capital letter.)

```
class Dog:
    pass
```

Instance Attributes:

- All classes create objects, and all objects contain characteristics called attributes (referred to as properties in the opening paragraph). Use the __init__() method to initialize (e.g., specify) an object's initial attributes by giving them their default value (or state). This method must have at least one argument as well as the self variable, which refers to the object itself (e.g., Dog).

```
class Dog:
    # Initializer / Instance Attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Class Attributes

- While instance attributes are specific to each object, class attributes are the same for all instances—which in this case is *all* dogs.

```
class Dog:
    # Class Attribute
    species = 'mammal'

    # Initializer / Instance Attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Instantiating Objects:

```
class Dog:

    # Class Attribute
    species = 'mammal'

    # Initializer / Instance Attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age


# Instantiate the Dog object
philo = Dog("Philo", 5)
mikey = Dog("Mikey", 6)

# Access the instance attributes
print("{} is {} and {} is {}.".format(philo.name, philo.age, mikey.name,
                                                                mikey.age))

# Is Philo a mammal?
if philo.species == "mammal":
    print("{0} is a {1}!".format(philo.name, philo.species))
```

Instance Methods:

- Instance methods are defined inside a class and are used to get the contents of an instance. They can also be used to perform operations with the attributes of our objects. Like the __init__ method.

```
class Dog:

    # Class Attribute
    species = 'mammal'

    # Initializer / Instance Attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def description(self):
        return "{} is {} years old".format(self.name, self.age)

    # instance method
    def speak(self, sound):
        return "{} says {}".format(self.name, sound)

# Instantiate the Dog object
mikey = Dog("Mikey", 6)

# call our instance methods
print(mikey.description())
```

```
print(mikey.speak("Gruff Gruff"))
```

Inheritance in Python:

```
# Parent class
class Dog:

    # Class attribute
    species = 'mammal'

    # Initializer / Instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def description(self):
        return "{} is {} years old".format(self.name, self.age)

    # instance method
    def speak(self, sound):
        return "{} says {}".format(self.name, sound)


# Child class (inherits from Dog class)
class RussellTerrier(Dog):
    def run(self, speed):
        return "{} runs {}".format(self.name, speed)


# Child class (inherits from Dog class)
class Bulldog(Dog):
    def run(self, speed):
        return "{} runs {}".format(self.name, speed)


# Child classes inherit attributes and
# behaviors from the parent class
jim = Bulldog("Jim", 12)
print(jim.description())

# Child classes have specific attributes
# and behaviors as well
print(jim.run("slowly"))
```

Q1.

Write a Python program that makes the empirical analysis of the insertion-sort algorithm;

- Writes functions to sort the list according to the insertion-sort algorithm, which is given below.

ALGORITHM *InsertionSort*$(A[0..n-1])$

```
//Sorts a given array by insertion sort
//Input: An array A[0..n − 1] of n orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```

- Program generates 4 different random arrays with the size 1000. The numbers will be between 0-100000.

Output:

```
Some part the array 1
[380170, 484134, 331676, 630373, 343659, 621561, 891107, 158343, 787461, 902775]
 0.0781226 seconds

Some part the array 2
[48582, 472950, 155389, 906021, 676973, 626380, 661377, 528715, 320457, 473280]
 0.0781069 seconds

Some part the array 3
[198872, 844114, 207122, 473966, 125351, 220900, 392492, 242143, 133899, 835626]
 0.0624971 seconds

Some part the array 4
[373808, 563686, 989946, 525684, 953535, 438913, 564173, 623172, 995086, 124992]
 0.0624962 seconds
```

Q2.

Topological Sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. A topological ordering is possible if and only if the graph has no directed cycles, that is, if it is a directed acyclic graph (DAG). Any DAG has at least one topological ordering.

Algorithm: The following algorithm will be applied in object oriented style.

Step 1: Create the graph as dictionary by calling addEdge (a,b).

Step 2: Call the topologicalSort( )

Step 2.1: Create a stack and a boolean array named as visited[ ];

Step 2.2: Mark all the vertices as not visited i.e. initialize visited[ ] with 'false' value.

Step 2.3: Call the recursive helper function topologicalSortUtil() to store Topological Sort starting from all vertices one by one.

Step 3: topologicalSortUtil(int v, bool visited[], int stack[]):

Step 3.1: Mark the current node as visited.

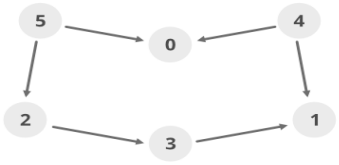
Step 3.2: Recur for all the vertices adjacent to this vertex if they are not visited.

Step 3.3: Push current vertex to stack which stores result.

Step 4: At last after return from the utility function, print contents of stack.

Write a Python Object-Oriented program that makes the topological sort using the given algorithm. Prints the final content of the stack.

REMINDER FOR TOPOLOGICAL SORT;



Adja cent list (G)
0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 2, 0

visited

0	1	2	3	4	5
false	false	false	false	false	false

Stack(empty)

Step 1: Topological Sort(0), visited[0] = true

↓

List is empty. No more recursion call.

Stack | 0 |

Step 2: Topological Sort(1), visited[1] = true

↓

List is empty. No more recursion call.

Stack | 0 | 1 |

Step 3: Topological Sort(2), visited[2] = true

↓

Topological Sort(3), visited[3] = true

↓

'1' is already visited. No more recurrsion call

Stack | 0 | 1 | 3 | 2 |

Step 4: Topological Sort(4), visited[4] = true

↓

'0' , '1' are already visited. No more recurrsion call

Stack | 0 | 1 | 3 | 2 | 4 |

Step 5: Topological Sort(5), visited[5] = true

↓

'2' , '0' are already visited. No more recurrsion call

Stack | 0 | 1 | 3 | 2 | 4 | 5 |

Step 6: Print all elements of stack from top to bottom

Q3. Write a Python program that gets an unsorted array and integer n value. Finds and returns the n^{th} element of the array by using LumutoPartition Algorithm given below.

ALGORITHM *LomutoPartition(A[l..r])*
//Partitions subarray by Lomuto's algorithm using first element as pivot
//Input: A subarray *A*[*l..r*] of array *A*[0*..n* − 1], defined by its left and right
// indices *l* and *r (l* ≤ *r)*
//Output: Partition of *A*[*l..r*] and the new position of the pivot

```
p←A[l]
s ←l
for i ←l + 1 to r do
    if A[i]<p
        s ←s + 1
        swap(A[s], A[i])
swap(A[l], A[s])
return s
```

Output:

The following is the output for the array : [10, 7, 8, 9, 1, 5]

```
Enter the item number that will be searched as index value:3
Partially Sorted array: [1, 7, 8, 9, 10, 5]
3 th item of an array is:  9
```