Data Structures & Algorithms

Week 1

Introduction

Lecturer: Dr. Msagha J Mbogholi, PhD

Content

- Introduction
- Variables
- Abstract data types (ADT)
- Algorithms

Part 1

Introduction

Introduction

- Data, data, data….
- Is it not fair to state that the world currently revolves around data? Coming closer to home, isn't every decision you make based on some kind of information (data) that your mind has already processed? Think about it.
- Our brains have been programmed from an early age to process what is happening around us and draw conclusions from it. Look at a child from early age, they cry out loudly when they are hungry, sleep when satisfied, cry again when uncomfortable (like when they need a change of clothes), and so on…these you may argue are inbuilt (or wired) from birth.
- However, as they grow older they learn some things the hard way (like fire burns when you put out your hand to it), and they learn other things by observing. Yet more is learnt from older people by being told to do or not to do certain things…suffice to say there are many ways by which children learn from infancy, to adolescence, to adulthood, to (even) old age!
- Your brain is constantly learning new things, sometimes without your conscious help. So how does the brain go about processing and learning?

Introduction (cont’d)

- The brain is actually the model on which all computers have been built. It receives input (from the environment), processes it, and then makes a decision on what to do thereafter (output). Let’s demonstrate this using a simple example:
- You have visited a lodge/camping site in the famous Masai Mara in Kenya, during the famous wildebeest migration. One night as you sleep you hear a lion roar nearby (too near for your comfort); how does the brain process this? Input = the lion roar; process = dangerous, not good; output/decision = wake up to make a more accurate assessment as to how much danger you’re in / release more adrenaline to the body. If you come face to face with the lion (without security) well, …..story for another day.
- What is the point here? We are demonstrating how the brain processes are part of the inspiration for this course, that I shall take you through in the next 12 weeks; so don’t get bored just yet.

Introduction

- Let’s revisit the lion situation we just imagined in the last slide. Clearly the brain has to have a way to differentiate all the different situations you find yourself in; when it’s something good it releases happy hormones like dopamine, when it’s exciting a release of adrenaline, and so on.
- This tells us that there is somewhere the brain processes information and makes decisions based on something…that something is mostly past experiences, information learnt from other sources such as books/people, inbuilt information in your DNA (debatable?), and reactions to yet unknown phenomena.
- This tells us that the brain has a way to receive data, organize it according to some system, store it accordingly, and associate it with various outputs/reactions. Moreover, even without some intimate knowledge of neuroscience we can tell that the brain is the most complex organ that is known to man in terms of its working.
- The brain has different storage areas in it, and different inputs are stored and classified accordingly. This introduces us to data structures and this course.

Introduction (cont'd)

- Just for your information: "Memories aren't stored in just one part of the brain. Different types are stored across different, interconnected brain regions. For explicit memories – which are about events that happened to you (episodic), as well as general facts and information (semantic) – there are three important areas of the brain: the hippocampus, the neocortex and the amygdala. Implicit memories, such as motor memories, rely on the basal ganglia and cerebellum. Short-term working memory relies most heavily on the prefrontal cortex." (https://qbi.uq.edu.au/brain-basics/memory/where-are-memories-stored#:~:text=Memories%20aren't%20stored%20in,across%20different%2C%20interconnected%20brain%20regions.)
- These are different parts of the brain that are structured differently according to the information that they hold. The modern day computers store data in much the same way that the brain does.
- Therefore, there are different types of "storage areas" that a programmer can use to store (and subsequently manipulate) data, depending on the type of data and what that data is to be used for.

Introduction (cont'd)

- Naturally a computer doesn't come equipped with these storage areas built for specific types of data and applications; it is the work of the programmer to create them.
- These storage areas are called data structures (and rightfully so). As the name implies they are the structures created to store and manipulate data to solve problems.
- This course is about learning when to use the data structures and how they work. It is not advisable to use one data structure for a type of data not applicable to it; it won't give the desired results and generally make it hard to manipulate the data.
- How about algorithms? Where do they fit in the greater scheme of things? Let us go back to examine why we write programs in the first place. You have learnt in other introduction to programming courses that we write programs to solve a problem or to exploit an opportunity.

Introduction (cont'd)

- However, problems aren't written in any programming language. The problem originates from a real world scenario and has to be converted into a programming language so that the application can understand it, and consequently solve it.
- In this course we shall learn how to understand the problem, write it down in plain English, convert it into a series of steps that will solve the problem, write it down in a language that is program independent, and then choose the programming language for implementation (and solving) the problem. This series of steps is what algorithmic design is all about.
- Thus we can now see the connection between data structures and algorithms:
- World problem → algorithm → (programming language → data structure)
- Data structures are declared differently in programming languages, according to the syntax and semantics of the chosen language. That is to say that a data structure created using Python will be declared differently in C++, or any other language. Nonetheless, once you are sure which structure you wish to use then the language implementation is not difficult.

Introduction (cont’d)

- In our introduction lesson we wish to discuss the following:
- Introduce some definitions and different data types used in computing
- Introduce algorithms and how they work
- Introduce a way of evaluating the effectiveness of an algorithm

Variables

2.1 Variable types

- As we know from our basic knowledge of the term variable, it is a value that keeps changing (from the word ‘to vary’, which means to change).
- The value of a variable at a given time is dependent on the information passed to it at that time.
- There are different types of variables, depending on the domain of application. There are variables associated with mathematics (mostly in statistics): nominal, ordinal, continuous, discrete and numeric. These are described in other courses such as “research methods and technical writing” and “machine learning”. These courses are available on this platform (simply search for the name of the course).
- Then we have variables that are associated with programming: constants, global, instance, local and class variables.
- Let us briefly describe these types of variables as we shall refer to them often during this course.

2.2 Definitions

- Constants: a constant is a value that does not change. In different programming languages a constant can be declared by a user, or it can be an in-built value that is called from a library using the language syntax.
- In-built constants: for example, in mathematics the value of pi used in calculations involving circles is a constant (3.14…). This means the value is stored in the language library (no need for you to keep typing 3.14 in every calculation). In Python the value is stored in the math library and so we have to import the library then call pi. For example let's say we wish to calculate the area of circle with radius = 7; we implement the code as follows:
- Import math
- radius = 7
- #area = 2 * pi * r*r, where r is the radius
- area = 2 * math.pi *radius *radius
- print(area)

2.2 Definitions

- User-defined constants: these are constants that are created and declared by the user. The value of a user defined constant can change in different programs, or it can be declared as of one definite value. Let's demonstrate this using both Python and Java:
- In Python, a variable is shown to be a constant by declaring it using capital letters. Essentially this means Python doesn't have the provision to declare a value or variable as constant. By using capital letters it is accepted that you are declaring a variable to be a constant; however, ensuring its value doesn't change in the program is entirely up to you. For example, TOP_VALUE = 100; means every time the variable TOP_VALUE is called, its value will be 100. In newer versions of Python the term 'final' can also be used. Nonetheless, Python has a data structure where all the values are 'read-only' (immutable) and can't be changed; this will be discussed more in a later lesson.
- In Java constants are declared using the 'static' and 'final' modifier. For example, static final double PI = 3.14; or final double PI = 3.14; or even final char b = 50. These are all valid declarations. Notice that whereas you can use upper or lowercase letters it is advisable to use uppercase, so that other programmers are aware that this variable is a constant.

2.2 Definitions (cont'd)

- Global variable: this is a variable that is available throughout the program. This means that it can be called from anywhere in the program. Let us examine how it is implemented:
- In Python a variable that is not declared within a function is a global variable; this means that any function can use this variable. Further, to make a variable global from within a function, use the global keyword. For example:
- example = 5
- # example is a global variable declared outside the function myfunc()
- def myfunc():

  print ("my global variable is" + example)
- def myfunc()

  global example

  example = 10
- myfunc()

print("my global variable value example in myfunc is:" + example

2.2 Definitions (cont'd)

- As can be seen the main purpose of having global variables is to be able to use them in different functions in a program.
- Local variable: a local variable is one which is available only to the function in which it is declared. This means that it can't be called from outside that function. Let us demonstrate this in Java this time using a simple example:
- Public class mylocalExample{
- Public double area1(double area){
- //local variable s
- double s = area * 4;
- return s }}
- In mylocalExample class we have declared a method called area1 which takes the given area and quadruples it; the result is stored in a local variable s. Therefore, s is only available to the method area1.

2.2 Definitions (cont'd)

- Class variable: a class variable is one that is available to the whole class in which it is declared. In Java it is normally declared under the class name, outside of all methods and constructors; further only one copy of the class variable per class. The same is true in Python. Let us declare a class variable using both languages.
- In Python:
- class myDemo:
- teacher_name = "Professor John"
- #teacher_name is a class variable and only one copy of it exists in the class
- # constructor and functions follow…some code
- In Java:
- Public class mydemo {
- static string teacher_name = "Professor John"

//teacher_name is declared a class variable using the static keyword; methods and //constructors follow after this. }

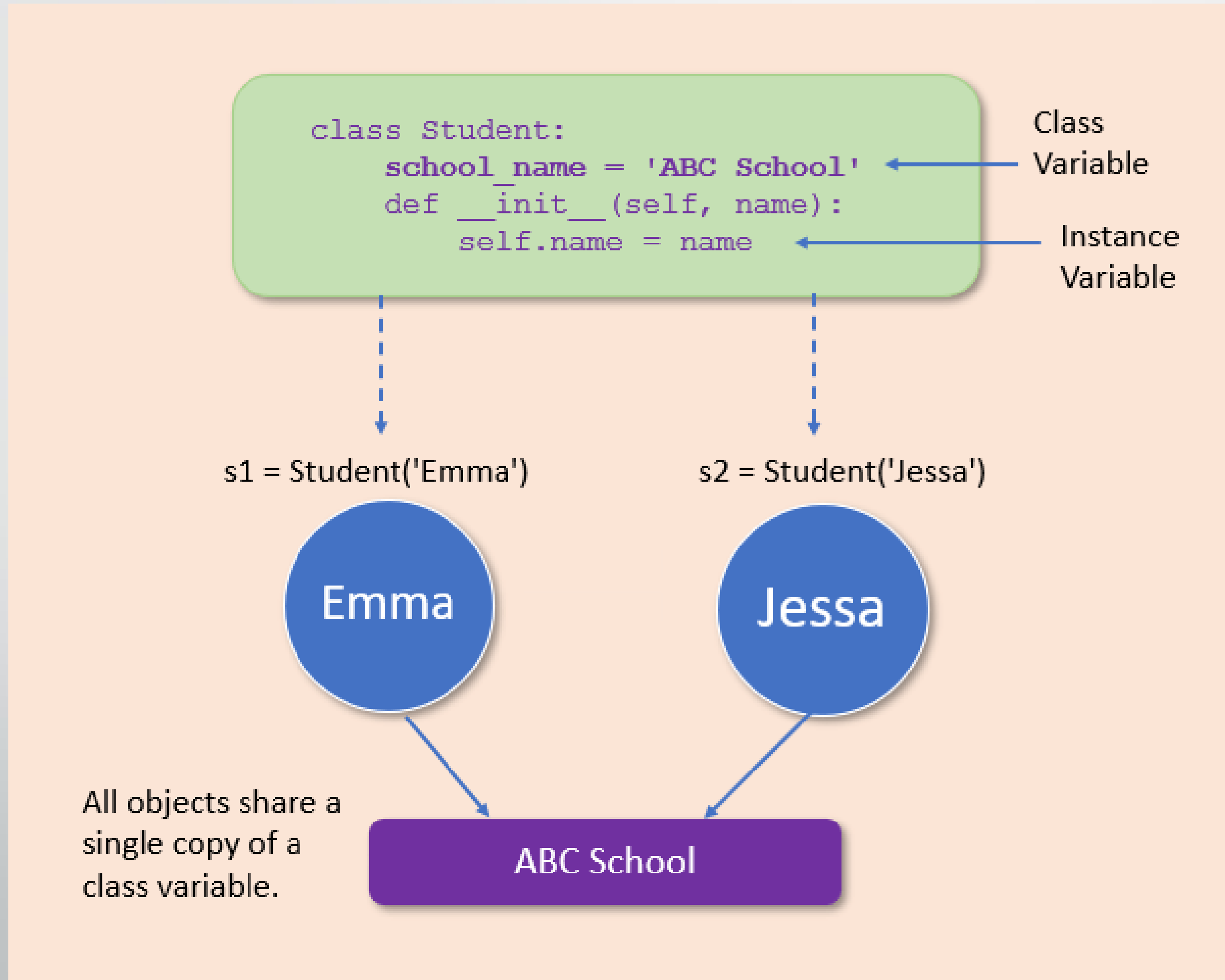
2.2 Definitions (cont'd)

- Instance variable: an instance variable is created when an object is instantiated (hence the name from the term to instantiate). Whenever a class is created it declares the characteristics of its members; this is because the class itself is simply a blueprint that declares the general characteristics of all its members. In simple terms you can apply methods to members of the class, what we call objects. Therefore the variables that an object creates are called its instance variables. Let's examine an example:
- In Python: instance variables are created within the constructor method of the class; Figure 1 demonstrates an example of the difference between class and instance variables. For example;

```
class Student:
    def__init__self(name, age):
        self.name = name
        self.age = age
new_student = Student ("Mike", 21)
#the parameters "Mike", and age "21" belong to new_student, which is an instance of the class Student.
```

Figure 1

Instance and Class Variables



(From Vishal, 2021)

2.2 Definitions (cont'd)

- Instance variable (cont'd):
- In Java: instance variables (unlike in Python) are declared outside of the constructor. Table 1 describes the differences between instance and class variables in Java. Let us demonstrate an instance variable using an example;
- Public class VariableTypeExample {

```
  int myVar;
  static int age = 30;
  public static void main (String args []) {
        VariableTypeExample myObj = new VariableTypeExample();
        myObj.myVar = 45;
        System.out.println("Value of instance variable:" + myObj.myVar);
        System.out.println("Value of class variable:" + VariableTypeExample.age); } }
```

- What is the output of this program?

Table 1

Class and Instance Variables in Java

Instance variables	Static (class) variables
Instance variables are declared in a class, but outside a method, constructor or any block.	Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.	Static variables are created when the program starts and destroyed when the program stops.
Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.	Static variables can be accessed by calling with the class name *ClassName.VariableName*.
Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.	There would only be one copy of each class variable per class, regardless of how many objects are created from it.

(From Ali, 2019)

Abstract Data Types

3.1 Introduction

- The Oxford dictionary defines abstraction as *inter alia*, “the process of considering something independently of its associations or attributes.”
- Abstraction is a concept we encounter in everyday life; only that we are not aware of it. Actually, by the mere fact that we aren’t aware of it tells us how effective it is. Abstraction is all about making our lives easier by sparing us the details of implementation of the devices we use.
- At a personal level we use abstraction quite unknowingly when we wish to indicate we will do something, while sparing the other party the details of how it will be done. For example a parent might tell a child they will pay their school fees; the child is not concerned how the fees will be paid, rather they just know their fees will be paid.
- Look at your mobile phone; when you wish to call a contact you only need to locate them in the phonebook, hit dial, and wait to be connected. In the case of voice assisted dialing you just say something like “Call Bryan” and the phone does the rest for you.

3.1 Introduction (cont'd)

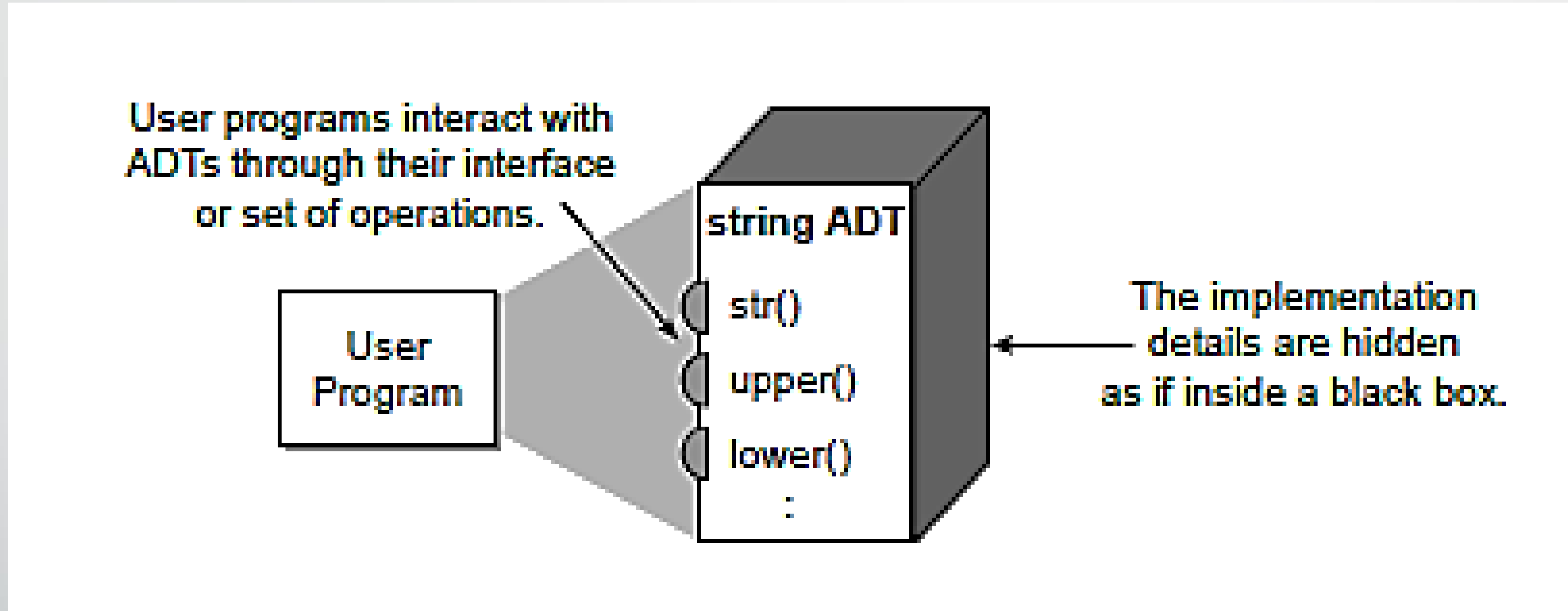
- But wait; have you considered how many processes are involved in making that simple phone call? Just to give you an idea; the phone software has associate the name with a phone number (just like DNS), then the dialing process is another story altogether....connect to the network, find a free line at the exchange, find the receiver (another complex process in itself), and so on. I think we get the idea.
- In computing an abstraction is "a mechanism for separating the properties of an object and restricting the focus to those relevant in the current context. The user of the abstraction does not have to understand all of the details in order to utilize the object, but only those relevant to the current task or problem." (Necaise, 2011)
- Necaise (2011) describes two types of abstraction: procedural abstraction, which is making use of a function or method without knowing how it implements the functionality (much like knowing how to make a call, without knowing the details of the process). Data abstraction is about separating "the properties of a data type (its values and operations) from the implementation of that data type."

3.2 Abstract data type (ADT)

- An abstract data type (or ADT) is a programmer-defined data type that specifies a set of data values and a collection of well-defined operations that can be performed on those values. (Necaise, 2011).
- As per the principle of abstraction these data types are defined independently of their implementation; thus, allowing us to focus on the use of the data type rather than how it is implemented.
- Interaction with the data type is through an interface or a defined set of operations (also known as encapsulation in object oriented programming). Consequently, abstraction allows us to place emphasis on the functionality of the ADT rather than the implementation of that functionality.
- The black box concept described in object oriented programming illustrates the working of an ADT; the inside of the black box holds the implementation details while users only see the surface and interact with the ADT from there. Figure 2 illustrates this concept.

Figure 2

Black Box Implementation in ADT



(From Necaise, 2011)

3.2 Abstract data type (ADT)

- There are various operations that can be performed on instances of an ADT in this scenario. However, these can be grouped broadly into four categories as described by Necaise (2011): "
- Constructors: Create and initialize new instances of the ADT.
- Accessors: Return data contained in an instance without modifying it.
- Mutators: Modify the contents of an ADT instance.
- Iterators: Process individual data components sequentially."

3.2.1 Advantages

- Necaise (2011) describes several advantages of working with ADTs using the black box method (focusing on the 'what' rather than the 'how'):
- We can focus on solving the problem at hand instead of getting bogged down in the implementation details.
- We can reduce logical errors that can occur from accidental misuse of storage structures and data types by preventing direct access to the implementation.
- The implementation of the abstract data type can be changed without having to modify the program code that uses the ADT. There are many times when we discover the initial implementation of an ADT is not the most efficient or we need the data organized in a different way.
- It's easier to manage and divide larger programs into smaller modules, allowing different members of a team to work on the separate modules. Large programming projects are commonly developed by teams of programmers in which the workload is divided among the members. By working with ADTs and agreeing on their definition, the team can better ensure the individual modules will work together when all the pieces are combined.

3.3 Data structures

- In some programming languages (such as Python) implementation of the ADTs is provided in the libraries. However, there are certain times where the programmer (you that is) will have to define and create your own ADT; this means that you will then have to provide the implementation details yourself.
- ADTs can be categorized as being either simple or complex. A simple ADT has one or perhaps a few individually named data fields. A complex ADT is one which has a collection of data items / values.
- The complex ADTs are the subject of our course; how individual data elements are stored, organized and manipulated provides the strengths and characteristics of individual data structures.
- There are many common data structures such as lists, stacks, arrays, dictionaries, trees, sets and so on. All data structures store a collection of values, but differ in how they organize the individual data items and by what operations can be applied to manage the collection. The choice of a particular data structure depends on the ADT and the problem at hand. Some data structures are better suited to particular problems. (Necaise, 2011)

3.3.1 Definitions

- Let us define some common terms used in this domain. Whereas most of these definitions can be found in myriad sources online, we shall use the definitions provided by Necaise (2011):”
- A collection is a group of values with no implied organization or relationship between the individual values. Sometimes we may restrict the elements to a specific data type such as a collection of integers or floating-point values.
- A container is any data structure or abstract data type that stores and organizes a collection. The individual values of the collection are known as elements of the container and a container with no elements is said to be empty.
- A sequence is a container in which the elements are arranged in linear order from front to back, with each element accessible by position.
- A sorted sequence is one in which the position of the elements is based on a prescribed relationship between each element and its successor. For example, we can create a sorted sequence of integers in which the elements are arranged in ascending or increasing order from smallest to largest value.”

3.4 Bag ADT

- In most instances when working with a list, most programmers find that it provides the most basic of operations required for manipulation, storage and organizing of its elements. However, there are several scenarios where we use an ADT called a bag.
- We briefly describe the bag in this lesson as it is not covered in detail elsewhere in this course.
- A bag is a container that stores a collection in which duplicate values are allowed. The items, each of which is individually stored, have no particular order but they must be comparable. In Python the functions that are used with a bag are *bag()*, *length()*, *contains(*item), *add* (item), *remove*(item), and *iterator().*
- Bags can be created and used in both Java and Python. To understand this better let us consider a simple code snippet of a bag implementation in Python.

3.4 Bag ADT (cont'd)

```
myexampleBag = Bag()
myexampleBag.add(15)
myexampleBag.add(7)
myexampleBag.add(29)
myexampleBag.add(1)
present = int( input("Guess a number contained in the bag.") )
if present in myexampleBag:
    print( "The bag contains the number", present )
else :
    print( "That number is not in the bag", present )
```

3.4 Bag ADT (cont'd)

- As described earlier, the choice of collection depends on the type of data involved, how it is to be stored, and what operations need to be performed for data manipulation.
- In the case of the bag, it is used in place of a list in most instances where there are large programs and multiple team members. Necaise (2011) describes these instances: "…By working with the abstraction of a bag, we can: a) focus on solving the problem at hand instead of worrying about the implementation of the container, b) reduce the chance of introducing errors from misuse of the list since it provides additional operations that are not appropriate for a bag, c) provide better coordination between different modules and designers, and d) easily swap out our current implementation of the Bag ADT for a different, possibly more efficient, version later."
- Nonetheless, the bag ADT collection can also be implemented using a list or dictionary data structure depending on the situation.

3.4 Bag ADT (cont'd)

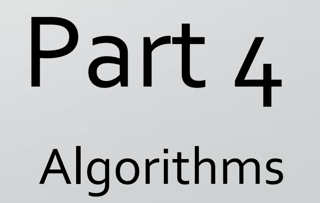
- Consider the situation where it is desirable to implement the bag ADT using a list. In order to ascertain whether the list implementation will suit the requirements, an examination of how each bag operation will be implemented by the list is carried out. Necaise (2011) does the check as follows: "
- An empty bag can be represented by an empty list.
- The size of the bag can be determined by the size of the list.
- Determining if the bag contains a specific item can be done using the equivalent list operation.
- When a new item is added to the bag, it can be appended to the end of the list since there is no specific ordering of the items in a bag.
- Removing an item from the bag can also be handled by the equivalent list operation.
- The items in a list can be traversed using a for loop and Python provides for user-defined iterators that be used with a bag."
- Thus in this case (for this particular itemized list) each bag operation can be implemented using the functionality of a list; thus it will suffice.

3.5 Criteria

- We have seen that the number of data structures are many, providing different functionalities for the complex ADT in question. All data structures provide different functionalities for storing, organizing and manipulating data. The big question therefore, is which one to use in a given situation? Necaise (2011) offers three criteria to use in the choice of data structure to use: "
- Does the data structure provide for the storage requirements as specified by the domain of the ADT? Abstract data types are defined to work with a specific domain of data values. The data structure we choose must be capable of storing all possible values in that domain, taking into consideration any restrictions or limitations placed on the individual items.
- Does the data structure provide the necessary data access and manipulation functionality to fully implement the ADT? The functionality of an abstract data type is provided through its defined set of operations. The data structure must allow for a full and correct implementation of the ADT without having to violate the abstraction principle by exposing the implementation details to the user. ..(cont'd)

3.5 Criteria (cont’d)

- …(cont’d)
- Does the data structure lend itself to an efficient implementation of the operations? An important goal in the implementation of an abstract data type is to provide an efficient solution. Some data structures allow for a more efficient implementation than others, but not every data structure is suitable for implementing every ADT. Efficiency considerations can help to select the best structure from among multiple candidates.”



Part 4

Algorithms

4.1 Introduction

- In this section we describe what an algorithm is and how it is related to this course. Recall that the course title is “data structures and algorithms”?
- An algorithm is a simple step by step unambiguous (or clear if you like) way of solving a problem. The key to this definition is that the steps must be performed in a specific order (step by step) and they must be finite (countable).
- We use algorithms in solving our day to day problems: going to work, returning home, preparing for exams, and so on.
- We do realize of course that there are many ways to solving a problem; however, would it not be more practical to solve problems in the easiest and most efficient manner? This saves on time, resources used, energy, and so on.
- How does this relate to data structures you may wonder? The answer to this in a bit.

4.2 Designing an algorithm

- Let us consider a way to prepare for an exam:
- Collect all the notes for the course.
- Do you have all the notes?
  - If yes, continue
  - Otherwise, look for notes from classmates
- Make a revision timetable
- Does the revision timetable include all the topics?
  - If yes, continue
  - If no, allocate more time to cover all topics
- Fill in the topics for each time slot
- .....

This algorithm is one of many that can help a student prepare for an exam

4.2 Designing an algorithm (cont'd)

- Of course we do realize that there are many ways to prepare for an exam. For example one might decide to team up with others in order to prepare for the exam, or might decide to prepare using past exams, and so on.
- This shows us that there are many ways to solve a problem. As Karumanchi (2017) describes it, "In the traditional study of algorithms, there are two main criteria for judging the merits of algorithms: correctness (does the algorithm give solution to the problem in a finite number of steps?) and efficiency (how much resources (in terms of memory and time) does it take to execute the(m))."
- In the study of data structures we see that there are many ways of solving the same problems; thus it is of interest to us to know which is the most efficient way(s) of solving these problems.
- For example there are many ways to sort a given list, but which is the most efficient? This is where the connection between data structures and algorithms come in; we have to find a way to efficiently evaluate the different algorithms used by different data structures in order to know their efficiency (more of this in the next section of this lesson).

4.3 Algorithm analysis

- The goal of the *analysis of algorithms* is to compare algorithms (or solutions) mainly in terms of running time but also in terms of other factors (e.g., memory, developer effort, etc.) (Karumanchi, 2017)
- When running time analysis is being performed we mean how the algorithm behaves in terms of time as the size of the problem increases. The term input size is used to describe the number of elements (can be of any type) in the input. Some of the common types of inputs include size of an array (how increasing the size affects performance of the algorithm), polynomial degree (increasing powers of 2 for example), number of bits in binary representation, and so on.
- Note that there are other factors that are considered in running time analysis such as storage, memory, developer effort, and so on). However, running time remains the main parameter of measure in running time analysis.

4.4 Algorithm comparison

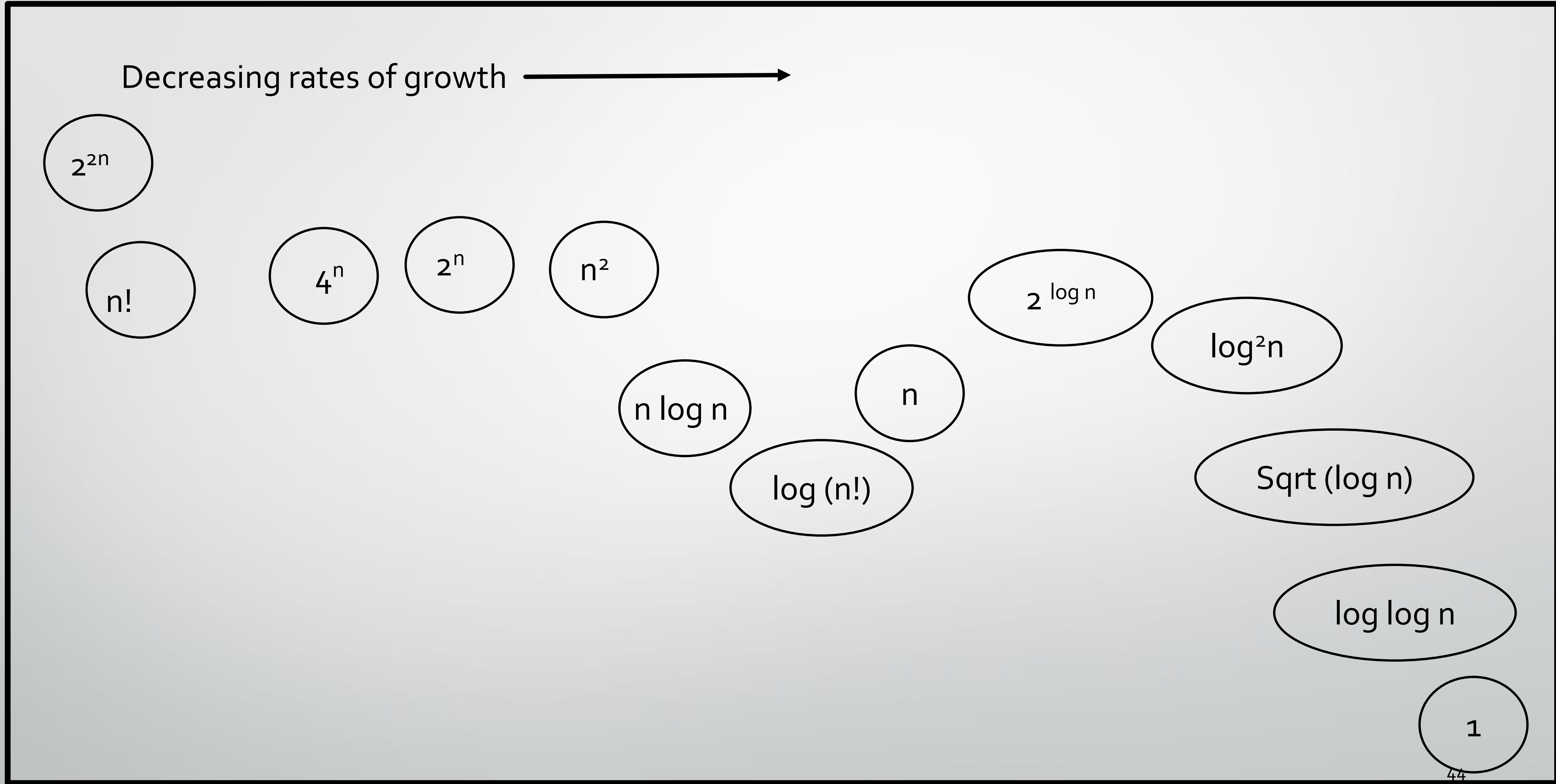
- How do we compare the efficiency of algorithms objectively? Of course there are many parameters but most fail the objectivity test due to one bias or another. Karumanchi (2017) dismisses two such measures thus: "
- **Execution times?** *Not a good measure* as execution times are specific to a particular computer.
- **Number of statements executed?** *Not a good measure*, since the number of statements varies with the programming language as well as the style of the individual programmer."
- Thus the best measure to use is to express the running time of the algorithm as a function of the input size n (this is what is used in all literature regarding this topic), and compare these different functions corresponding to running times. This then is a truly objective measure independent of factors such as the ones described above.

4.5 Rate of growth

- "Rate of growth is defined as the rate at which the running time of the algorithm is increased when the input size is increased. The growth rate could be categorized into two types: linear and exponential. If the algorithm is increased in a linear way with an increasing in input size, it is **linear growth rate**. And if the running time of the algorithm is increased exponentially with the increase in input size, it is **exponential growth rate**." (*Design and Analysis of Algorithm*, n.d.)
- Suppose you visited a restaurant and had dinner at a cost of say $30; you then had dessert of say $3. When your friend asks you tomorrow how much dinner is at that restaurant will you give him an approximate figure of 30 or 33? For most people they would just say it's about 30 dollars.
- The reason for this is that the $3 is very small in comparison to the $30 (plus this sounds like a more round figure for approximation purposes).
- We can apply the same logic to functions; for functions with higher orders we can ignore the lower orders as input size increases since they become negligent values comparatively. Let's use the example of the equation $n^4 + n^2 + n + 200$; in this equation we can approximate it to n^4 for large n, since the other parameters will be very small compared to this one. Figure 3 compares different rates of growth.

Figure 3

Comparison of Rates of Growth



(Adapted from Karumanchi, 2017)

4.6 Types of analysis

- Karumanchi (2017) describes the different types of analysis of algorithms as follows: "To analyze the given algorithm, we need to know with which inputs the algorithm takes less time (performing well) and with which inputs the algorithm takes a long time. We have already seen that an algorithm can be represented in the form of an expression. That means we represent the algorithm with multiple expressions: one for the case where it takes less time and another for the case where it takes more time.
- In general, the first case is called the *best case* and the second case is called the *worst case* for the algorithm. To analyze an algorithm we need some kind of syntax, and that forms the base for asymptotic analysis/notation. There are three types of analysis:
- **Worst case:**
  - Defines the input for which the algorithm takes a long time (slowest time to complete).
  - Input is the one for which the algorithm runs the slowest.

4.6 Types of analysis (cont'd)

- **Best case**
  - Defines the input for which the algorithm takes the least time (fastest time to complete).
  - Input is the one for which the algorithm runs the fastest.
- **Average case**
  - Provides a prediction about the running time of the algorithm.
  - Run the algorithm many times, using many different inputs that come from some distribution that generates these inputs, compute the total running time (by adding the individual times), and divide by the number of trials.
  - Assumes that the input is random."
- Thus Lower bound <= average time <= upper bound
- The best, average, and worst times for a given algorithm can thus be described using mathematical expressions

4.6 Types of analysis (cont'd)

- We have now introduced key terms related to the analysis of algorithms. In later lessons we shall implement these concepts in order to understand what we call the complexity of equations and code. Terms such as asymptotic notation, big O, and so on shall be covered at that point.

Summary

- Variables that are associated with programming include constants, global, instance, local and class variables.
- A class variable is one that is available to the whole class in which it is declared. In Java it is normally declared under the class name, outside of all methods and constructors; further only one copy of the class variable per class. The same is true in Python
- An abstract data type (or ADT) is a programmer-defined data type that specifies a set of data values and a collection of well-defined operations that can be performed on those values.
- All data structures store a collection of values, but differ in how they organize the individual data items and by what operations can be applied to manage the collection. The choice of a particular data structure depends on the ADT and the problem at hand.
- In the traditional study of algorithms, there are two main criteria for judging the merits of algorithms: correctness (does the algorithm give solution to the problem in a finite number of steps?) and efficiency (how much resources (in terms of memory and time) does it take to execute them).
- Three types of analysis of algorithms in terms of time: worst case, average case, and best case. These are all expressed in mathematical notation based on the algorithm under investigation.

References

- Ali, J. (2019, July 30). *Object Oriented Programming in Pyth: Class and Instance Variables*. DigitalOcean. https://www.digitalocean.com/community/tutorials/understanding-class-and-instance-variables-in-python-3
- *Design and Analysis of Algorithm*. (n.d.). Www.tutorialspoint.com; Tutorials Point (India) Private Ltd. Retrieved September 10, 2023, from https://www.tutorialspoint.com/design_and_analysis_of_algorithms/analysis_of_algorithms.htm#:~:text=Rate%20of%20growth%20is%20defined
- Karumanchi, N. (2017). *Data structures and algorithms made easy : to all my readers : concepts, problems, interview questions*. Careermonk Publications.
- Necaise, R. D. (2011). *Data structures and algorithms using Python*. John Wiley And Sons.
- Vishal. (2021, July 27). *Python Class Variables*. PYnative. https://pynative.com/python-class-variables/