## TYPES OF PROGRAMMING LANGUAGE

There are many different types of programming language. Here we will look ar some of thse types, their features ad why they might be used.

### The need for different paradigms

A paradigm is a way of thinking. We can apply different paradigms to how we program. A common paradigm in programming is imperative programming. In linguistics, the imperative mood means the language we use to give orders, for example: Sit down. Eat up. Open the box. These sentences are all imperative - they're giving orders. Imperative programming languages are those in which we tell the computer what to do; we tell it how to solve a problem. **Procedural** and **object-oriented programming** are imperative paradigms.

In procedural programming, we use the program to tell the computer the steps we want the computer to go through to solve a problem. An alternative approach is declarative programming. With declarative programming, we tell the computer the qualities the solution should have. A common example of

declarative programming is SQL (Structured Query Language), where we describe what results we want from a database query but don't need to explain how we to get them (This will be studied in the A-Level course in detail). There are a number of subtypes of declarative language, including **logic** and **functional programming**.

Some languages allow programming in multiple paradigms. Python, for example, can be used procedurally but also supports object-oriented programming and some functional programming.

You will need to know about object-oriented programming for this course and so we will examine it in more detail later.

A programming language is referred to as 'Turing Complete' if it can solve all the problems it has been proved computers can solve. Most programming languages across different paradigms are Turing Complete. We don't therefore have different programming paradigms because some problems can only be solved in a particular type, but rather because some problems are better suited to being solved in a particular paradigm. A lot of work has been done, for example, using logic programming for natural language processing. By defining a language by facts and rules, it is possible to get a computer to infer some meaning from the sentences we use.

At the end of this chapter is a revision section of all the main programming paradigms available in addition to the ones shown tin the key terms box in blue on the next page.

**KEY TERMS:**

**PROCEDURAL PROGRAMMING** A program where instructions are given in sequence; selection is used to decide what a program does and iteration dictates how many times it does it. In procedural programming, programs are broken down into key blocks called procedures and functions. Examples of procedural languages include BASIC, C and Pascal.

**LOGIC PROGRAMMING** Rather than stating what the program should do, in logic programming a problem is expressed as a set of facts (things that are always true) and rules (things that are true if particular facts are true). These facts and rules are then used to find a given goal. The most commonly used logic language is Prolog.

**FUNCTIONAL PROGRAMMING** A function, in mathematics, takes in a value or values and returns a value, for example:

double(4) would return 8 highestCommonFactor(36,24) would return 12

In functional programming, a description of the solution to a problem is built up through a collection of functions. Examples include Haskell and ML.

**EVENT-DRIVEN LANGUAGES**

A program that waits for events such as the clicking of the mouse or the press of a key on a keyboard.

When an event occurs, it is processed using a defined sequence of instructions called an event handler.

Useful for control programs where events such as readings from sensors are used to control devices.

Examples: Visual Basic, C++, Javascript

**MARK-UP LANGUAGES**

Special coding instructions are used to indicate style and layout of text and other elements.

Widely used for creating web pages on the Internet. HTML (Hypertext Transfer Protocol) is the most common mark-up language used for creating web pages. Is is a standard that used text and tags to control what is displayed on the user’s computers. The tags such as <h1> (a start tag) or </h1> (an end tag) delineate text items and effect how they are displayed. The browser software interprets these tags to display on the user’s computers,

Examples: HTML. XML, XHTML, ASP.

**VISUAL LANGUAGES**

Allow the programmer to manipulate objects visually on a form, setting their layout and properties. The underlying program code is automatically generated.

Used for creating Windows (GUI) applications.

Examples: Visual Basic, Visual C++, Delphi

### Object-Orientated Programming

In object-oriented programming, we represent the solution to a problem through objects.

Each object has attributes (sometimes referred to as properties) that are variables that store information about that object. It also has methods. Methods are actions an object can carry out. These are the equivalent to subroutines.

**Example**

In the exam, pseudocode is uses. **Methods** represented with the terms 'procedure' and 'function' to denote whether or not they return a value, but really they should be referred to as methods. Real languages have different approaches. Java, for example, uses the keyword 'void' if it doesn't return a value or the data type/object type returned if it does.

For example, the following method does not return a value:

Public procedure changeColums(newVol)

Volume = newVol

endprocedure

The following method does return a value:

Public function getVolume()

Return colume

endfunction

### Classes and Objects

We can think of a class as a template. It defines what attributes and methods an object should have. It is the equivalent to a biscuit cutter, with our objects being the biscuits themselves. One of the benefits of

object-oriented programming is that once a class has been written it can be reused in other programs.

class Monster

private poisonous

private strength

private name

public procedure new(givenPoisonous, givenStrength, givenName)

poisonous=givenPoisonous

strength=givenStrength

name=givenName

endprocedure

public procedure eat()

print(name+" eats a hero. Mmmmmm Delicious!");

endprocedure

public procedure sleep()

print("Snore, Snore, Snore")

endprocedure

endclass

This class tells us that all objects of type Monster have the attributes poisonous, strength and name and the methods eat and sleep. The section starting public procedure new( ... is what is called a constructor. It describes what happens when an object of this type is created. In this case, it uses the values of the parameters passed to it to set the monster's attributes. In the main program we can have the lines:

monsterOne new Monster(true, 5, "Alvin")

monsterTwo = new Monster(false, 7, "wilfred")

The objects monsterOne and monsterTwo are created. Monster one is poisonous, has a strength of 5 and the name Alvin. Monster two is not poisonous, has a strength of 7 and the name Wilfred.

We can then use the method eat():

MonsterOne.eat()

This would cause the following to be displayed:

Alvin eats a hero. Mmmmmm Delicious!

### Inheritance

Often we will need classes that have similarities to another class but also their own distinct differences, for example in a company, all employees might have a salary, date of joining and email address. Different categories of employee might have additional attributes. A manager might have the additional attribute department. An engineer might have the additional method repair.

Inheritance allows us to create a class that has all the methods and attributes of another class as well as attributes and methods of its own. Going back to our example of Monster, let's create a new class Vampire.

class Vampire inherits Monster

endclass

Notice how the class line uses 'inherits'. This keyword tells us that Vampire has all the methods and attributes of Monster. (The pseudocode you will see in the exam will use the keyword inherits; real languages have different alternatives. Java uses extends, C# and c++ use a colon: . They all function in the same way.) We refer to Monster as the super (or parent) class and Vampire as the sub (or child) class.

At this stage, we could create objects of type Vampire but they would be exactly the same as objects of type Monster. We want Vampire to have the attribute hasCastle (as to whether or not they own a castle)

and the additional method drinkBlood.

Class Vampire inherits Monster

hasCastle=true

public procedure drinkBlood()

print(name+", the vampire, drinks the hero's blood")

endprocedure

endclass

If we write the code in the main part of the program:

vampireOne=new Vampire (false, 10, "Dracula")

a new Vampire is created, using the constructor from Monster. We can now use the method drinkBlood:

vampireOne. drinkBlood( )

Likewise, we can still do:

vampireOne. sleep ( )

Vampires don't tend to snore when they sleep (because they don't breathe). We therefore want the sleep method for a Vampire to be different. We can do this by overriding the Monster's sleep method. Overriding is when a method in a subclass is used to replace a method inherited from the super class.

class vampire inherits Monster

hasCastle=true

public procedure drinkBlood()

print(name+", the vampire, drinks the hero's blood")

endprocedure

public procedure sleep()

print("The vampire sleeps silently")

endprocedure

endclass

Now:

vampireOne.Sleep()

will display

The vampire sleeps silently

It would be better in this case if Vampire had its own constructor. This would allow us to set a starting value for hasCastle. Also, as no vampires are poisonous we don't need to take in a value for poisonous when creating a new vampire. To do this, we override the superclass's (Monster) constructor. In overriding it we still, in this case, want to use the superclass constructor. We can do this with the keyword super.

(Note this keyword can be used to call any other methods from the superclass too.)

class vampire inherits Monster

hasCastle=true

public new(givenHasCastle, givenStrength, givenName)

hasCastle=givenHasCastle

super.new(false, givenStrength, givenName)

endprocedure

…

…

endclass

We can now give Dracula a castle, creating him in the following way:

vampireOne=new Vampire(true, 10, "Dracula")

### Polymorphism

The word 'polymorphism' comes from the Greek meaning 'many forms'. You may well have come across polymorphism, depending on the programming language you have used, without realising it.

Consider the following code:

a="Hel"

b="lo"

c=a+b

print(c)

Now compare it with:

a=l

b=2

c=a+b

print(c)

In both cases we use the + symbol, but in each case it has different meanings. In the first example, + means concatenate as it is being used with two strings. In the second it means add these two numbers together, as it is being used with two integers. In other words, + has different forms according to its context.

Let's assume I want a monster zoo, which I am going to store in an array. There are going to be all sorts of monsters in this array but if my array is of type Monster, I can store all subclasses of Monster (Vampire, Goblin, and so on) in there. The technical term for this is a 'polymorphic array'.

Now I have this array I may wish to iterate through it and send all my monsters to sleep. Some monsters will have different sleep methods (for example we overrode the Vampire sleep method in the last section). This is no problem as polymorphism means (just as with the + in our example earlier) the correct sleep method will be called depending on the object type.

monsterA=new Goblin(23, false, 7, "Frank")

zoo[O]=monsterA

monsterB=new Monster(true,8, "Medusa")

zoo[l]=monsterB

monsterC=new Vampire(true, 10, "Dracula")

zoo[2]=monsterC

for i=O to 2

zoo[i].sleep()

next i

### Encapsulation

Imagine you have written a class called Airplane that is used as part of a program to calculate the fuel necessary for a flight and that this class has the attributes passengers, cargoWeight and fuel. What could go wrong if other classes had direct access to these attributes and could change them freely?

One possibility is that a weight is assigned that is too heavy for the plane to carry.

plane=new Airplane()

plane.weight=99999

It might be that the weight is updated but no code is run to update the fuel to take into account the new weight. More passengers could be added, which would add to the weight and fuel needed but these too might not be updated.

This is the sort of situation we wish to avoid. To do this we use ***encapsulation***.

Encapsulation is the pattern of making attributes in a class private but allowing them to be changed and accessed through public methods.

The keyword private means that the method or attribute following it is only accessible from within that class. If the Airplane class had the weight as private then any attempt to change it outside the class would result in an error.

**Airplane class:**

class Airplane

private weight

private fuel

private passengers

…

**Main Program:**

plane=new Airplane()

plane.weight=99999 <- *this line would cause an error*

We then provide a method to change the attribute and make this public. As the method is in the same class as the attribute, it is able to change it. By only allowing access via this method, the attribute can only be changed in the way we specify, for example:

class Airplane

private weight

private fuel

private passengers

public procedure setweight(enteredWeight)

if enteredWeight>maxWeight then

print("TOO heavy")

else

weight=enteredWeight

updateFuel( )

PRACTICE QUESTION:

Using the Monster class you made earlier, use encapsulation to ensure

the strength can only be set to a value between one and twenty.

endif

endprocedure

public function getWeight()

return weight

endfunction

endclass

**Main Program:**

plane=new Airplane()

plane.setWeight(500)

Typically when using encapsulation, each attribute will have a 'get' method (for example getweight), sometimes called the accessor, which allows other classes to see the value of an attribute and a set method (for example setWeight), and sometimes called the mutator, which allows the attribute value to be changed.

It should be remembered that encapsulation isn't there to stop malicious attempts to change attributes. It is there to reduce the chance of mistakes occurring through attributes being altered in an unforeseen way by other objects (which may well have been coded by the same person who coded the encapsulated class).

### Levels of Computer Language

Computers can only execute programs in binary machine code. In the beginning....programmers had to write programs in machine code...

A machine code program would look like..

00101000

10110011

10010101

01001011

etc...

Programming was difficult and tracking down errors (debugging) must have been a nightmare!

Each line of machine code consists of an instruction (opcode) that may by followed by an item of data (operand). This is then executed during a cycle of the fetch-decode-execute cycle.

Assembly Language was developed which gave mnemonics (meaningful abbreviations) to the machine code instructions.

Assembly language programs look more readable...

MOV AL,5

CLC

MOV AH,2

INT 21H

etc...

..but only just!

Programming languages are classified as **Low level** (near to the language a computer uses - ie machine code) or **High level** (near to the language a human uses).

Low level (Assembly) languages are:

* computer orientated
* difficult for programmers to develop code and to test and modify it. Simple tasks need a lot of instructions.
* used to develop programs that need to run very fast.
* Develop programs that take of the minimum amount of space
* These programs tend to be hard coded onto chips
* These programs tend to be in full control of the CPU

e.g. low level programs are used in everyday items such as a microwave, toaster, central heating systems, TVs, DVD players etc.

High level languages started to be developed in the mid 1950s - to make it easier for programmers to develop complex programs. They are...

* problem orientated
* easier for humans to understand and use.
* used to develop programs that can run on different computers.

**Examples of high-level languages are :**

* FORTRAN (FORmula TRANslator); mainly used for engineering/scientific computing.
* COBOL(COmmon Business Orientated Language). Still one of the main commercial data processing languages.
* ALGOL (ALGOrithmic Language) was also developed in the 1950s and many of the languages such as C and C++ are developed from it.
* BASIC (Beginners All-purpose Symbolic Instruction Code) was developed in the 1960s as a simple programming language.
* PASCAL was developed in the 1970s as a well-structured teaching language.
* JAVA - a recent language developed from C++; an object-oriented language useful for developing programs which work over the Internet.
* Visual Basic – developed in the early 1990, similar to Pascal and mainly DOS based which then developed in the mid 1990 as an IDE based package for creating windows based forms etc.
* Vb.net – developed in the early 2000 and is currently still being developed all the time.
* Other high-level languages include : LISP, PROLOG, PL/1 , ADA

Scientific languages (like FORTRAN and ALGOL) would have high-precision mathematical functions, whereas commercial languages (like COBOL) would have powerful file handling facilities.

Scientific languages would be needed for e.g. weather forecasting, real-time control systems, etc

Commercial languages would be needed for e.g. information retrieval systems.

### Programming Paradigms Revision / Summary \*\*\* Study this in detail

**Procedural languages**

* Procedural languages are used in traditional programming based on algorithms or a logical step-by-step process for solving a problem
* They obey (ordered) instructions
* They carry out actions / calculations etc.
* A procedural programming language provides the programmer a way to define precisely each step when performing a task
* Allows tight control over the underlying operation of the hardware
* Used in (large complicated) programs where similar operations may be carried out at varying stages of the program execution

**Scripting Language**

* Set of commands understood by the applications software
* Usually embedded in another language and is used to control aspects of the software
* Usually a High-level programming language
* Can be interpreted not compiled
* Scripting languages provides the programmer a way to define precisely each step when performing a task
* Allows tight integration with existing programs or data
* Script embedded in (the HTML in) a web site to control graphics, etc.
* Script embedded in a web site to load / execute a file when clicked, etc.

**Non-Procedural languages**

* Non-procedural programming languages allow programmers to specify the results they want without specifying how to solve the problem
* Non-procedural languages are to do with rules / making queries
* Used in database interrogation where retrieving answers are more important than the exact steps required to calculate the result
* Artificial intelligence and modelling applications are often written in a non-procedural language

**Object Orientated Language**

* Uses objects and classes - include both data and associated processing
* Applies the principles of encapsulation, inheritance and polymorphism to aid programming
* Enables production of buttons / icons etc. - useful in a visual environment
* A class defines the methods and properties (data) for a group of similar objects
* Once an object is created, knowledge of its implementation is not necessary for its use.
* Objects control how other objects interacts with themselves, preventing other kinds of errors, e.g. a programmer cannot set the width of a window to -500
* In Visual Basic, the programmer places objects on forms. It is an event-driven language
* An event, e.g. click a command button, initiates a sequence of code to be executed
* Objects created using object oriented languages can easily be reused in other programs

**Special Purpose Language**

* Languages that were designed with a specific purpose in mind as opposed to a more general use language
* Might have essential / helpful features relevant to the application
* Are available for simulation, control etc.
* Very specialised with built in functions/abilities that lend themselves directly to solving the problem that the language was design to work on
* Used in:
  + Computer aided design
  + Artificial intelligence
  + Expert systems
  + Scientific applications
  + Games programming (DirectX etc)

**4th Generation Language**

* First generation programming languages created construct above the machine-code program
* Each subsequent generation represented a further distancing from the binary code that the computer hardware actually reads
* Some packages, e.g. Microsoft Access, have in-built programming capabilities. This allows the programmer to customise general purpose packages to exactly meet the needs of the business.
* Generally a very high level programming language (English syntax and grammar)
* Many features such as query, manipulation of data
* May have report generators and possibly application generators
* May attempt to produce natural language interface
* Requires less programming skill
* Would be useful in a database query / manipulation situation
* Often used in conjunction with end user applications to customise their operation without requiring highly developed and specialised programming skills

**Natural Languages**

* The user would not need to structure voice (or typed) input in any way - could communicate with the computer as if with another person
* A natural language interface would need very high processing power / very complex software
* Natural language used by most people is very ambiguous / imprecise / doesn't tend to conform to set grammar / slang is often used / English language is changing
* Symbolic languages are capable of interpreting and processing queries by sentences, e.g. calculating mathematical equations
* Allows the user to speak in their normal everyday language in order to interact with the computer
* Speak everyday commands, such as “Open the last document I used”

**Visual Programming Languages**

* High level programming language
* Particularly suitable for production of objects / buttons / icons, etc.
* Particularly suitable for developing in a GUI / graphics content / event driven environment (e.g. double-click > execute)
* May be easier to learn / more intuitive because visual / very good help / tools available

**Application packages that have programming capabilities**

* Additional functionality can be added without a programmer / buying another package / program
* Can customise the package / tailor to specific needs etc
* Requires less / no programming skill > more help is available in the package
* Is probably cheaper / quicker since most facilities are provided by the package
* Can import / export from / to other packages
* Is less likely to contain errors “bugs” / package has already been well tested
* Users are probably familiar with interface
* Programming might be restricted and have certain functionality unavailable in the package