Programming Principles in Python

Introduction

Samuel A.

Lecturer

Samuel Adebayo

sadebayo@belfastmet.ac.uk

WK 1 Lecture 1 Module Outline and Introduction

Outline

- Module Induction
- Background to module (the why)
- Introduction to Computer Programming
- Basics of programming
- Natural Language
- Translators
- Compilers
- Assemblers

Arrangements

Lectures

• Programming Principles in Python	Wk 23 – 38 (Wednesdays)	M-1-4-09

Arrangements

Lectures

• Programming Principles in Python	Wk 23 – 38 (Wednesdays)	M-1-4-09

Labs/Tutorials

• Codes/Tutorials/Codewithme	Wk 25 – 40 (Fridays)	M-1-4-05

Arrangements

Lectures

• Programming Principles in Python	Wk 23 – 38 (Wednesdays)	M-1-4-09

Labs/Tutorials

• Codes/Tutorials/Codewithme	Wk 25 – 40 (Fridays)	M-1-4-05

Both Lectures and Labs are compulsory.

PCs are available to code on.

Assessment- dates to be confirmed

The module is assessed through a continuous assessment which is made up of a coding portfolio, programming assessment, and Assignments.

Week	Assessment	Content	Weightning
4	Assignment 1	Wks 1 -4	10%
6	Assignment 2		15%
8	Assignment 3		20%
9	Class Tests		20%
10	Assignment 4		10%
13	Assignment 5		10%
	Project		15%

Requirements:

Attendance at Lectures and Labs.

8 Hours of private study per week including assessment.

Completion of all work as outlined.

Asks questions if you don’t understand, Teams, email etc.

Outline

- Module Induction
- **Background to module (the why)**
- Introduction to Computer Programming
- Basics of programming
- Natural Language
- Translators
- Compilers
- Assemblers

Why?

Programming is an essential skill in today's digital age, and studying it can lead to numerous benefits and opportunities:

- **Career Advancement**: Knowledge of programming can lead to lucrative careers in software development, data analysis, and many other technology-related fields.
- **Problem-Solving**: Programming provides a structure for solving complex problems through logic and algorithms.
- **Collaboration**: Programming is often a collaborative effort, allowing you to work with others and learn from their skills and perspectives.

Why?

Programming is an essential skill in today's digital age, and studying it can lead to numerous benefits and opportunities:

- **Creativity**: Programming allows you to turn your ideas into reality by creating software, websites, and applications.

- **Digital Literacy**: A basic understanding of programming is becoming increasingly important as technology continues to play a central role in society and commerce.

Overall, studying programming can open doors to new and exciting career paths, as well as develop valuable skills in logic, problem-solving, and creativity.

Outline

- Module Induction
- Background to module (the why)
- **Introduction to Computer Programming**
- Basics of programming
- Natural Language
- Translators
- Compilers
- Assemblers

Programming

- Programming is the process of designing, writing, testing, debugging, and maintaining the source code of the computer software.
- It involves the use of programming language to create instructions that a computer can understand and execute to perform specific tasks.
- The goal of programming is to develop software that solves problems, automates processes, and enhances human productivity.

Programming

- Programming also involves problem-solving, critical thinking, and creativity. Programmers work on a wide range of software applications, from simple scripts to complex systems, and from desktop applications to web and mobile apps.

- **Effective Programming!**
- **High Quality Software**
- **Scalable and Reliable**

Programming: what is required?

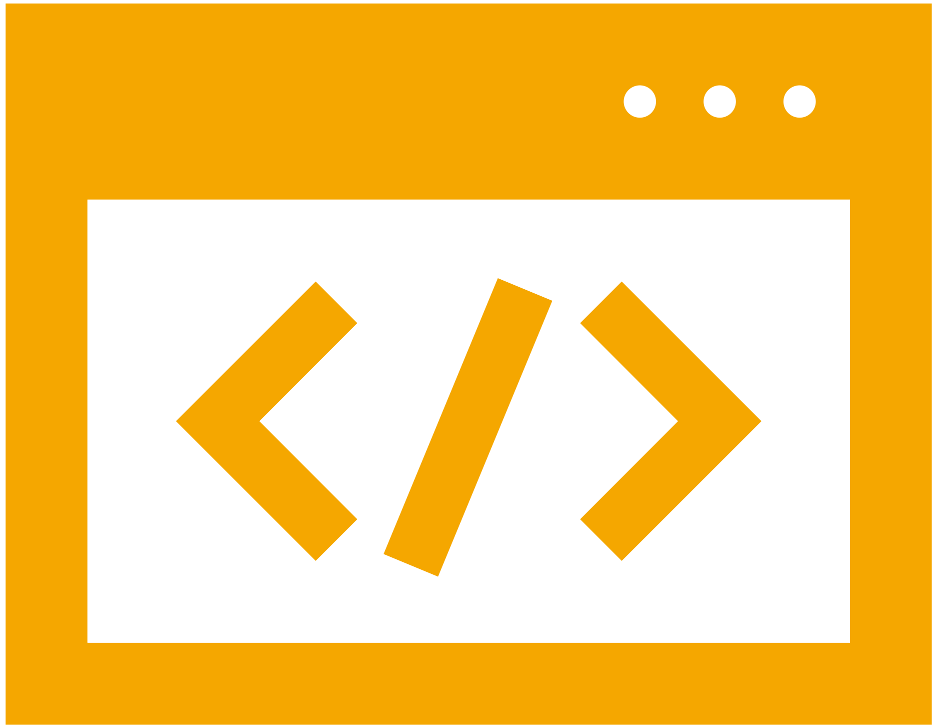
- Ability to understand and analyze the problem.
- Design a solution
- Write and test codes
- Debugging
- Maintainability

Types of Programming

- Imperative Programming
- Functional Programming
- Declarative Programming
- Event-driven Programming
- Scripting Programming
- **Object-oriented Programming**

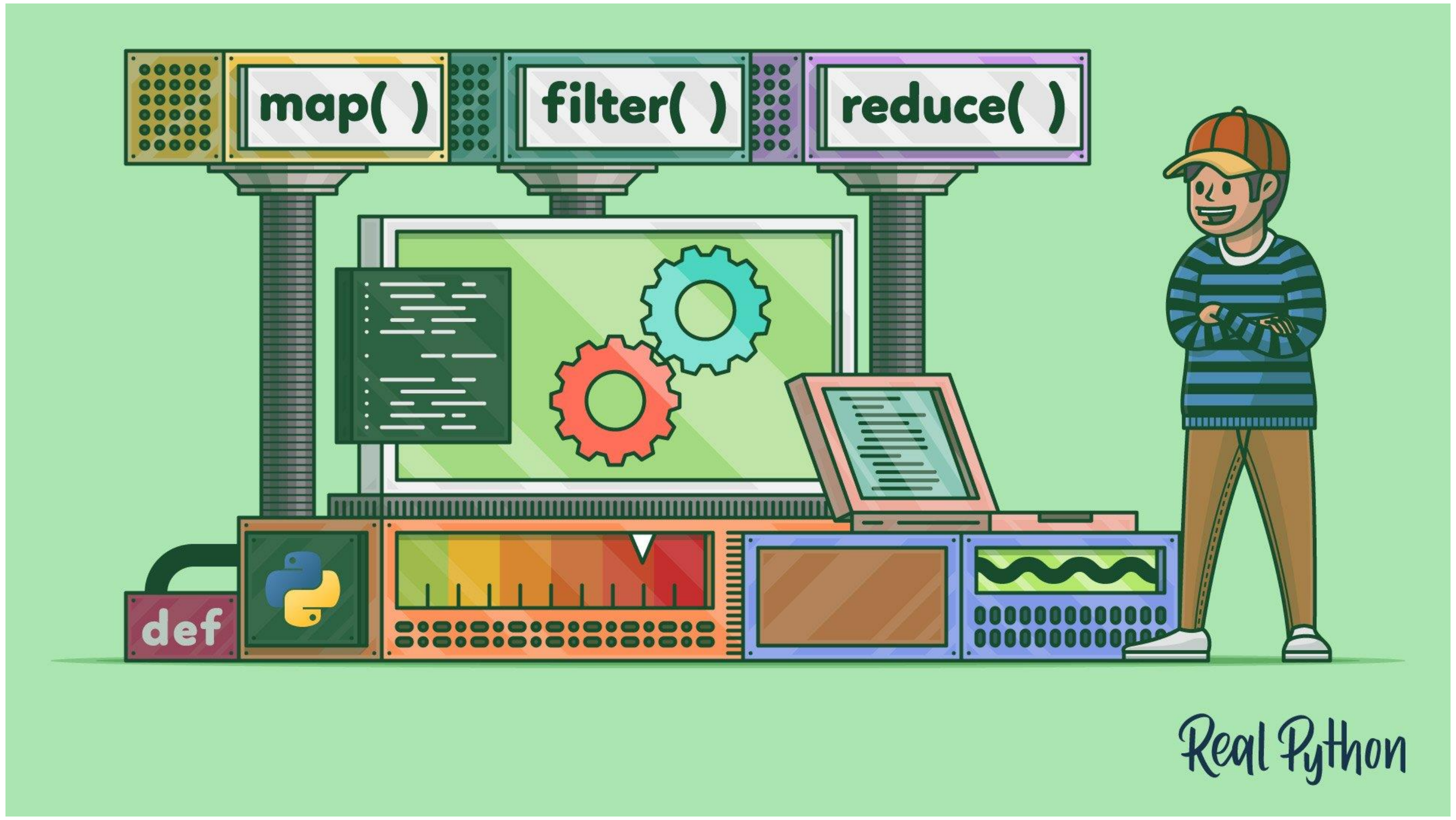
Imperative Programming

This type of programming is based on procedures and sequences of statements that change a program's state. It includes languages such as C, C++, and Pascal.



Functional Programming

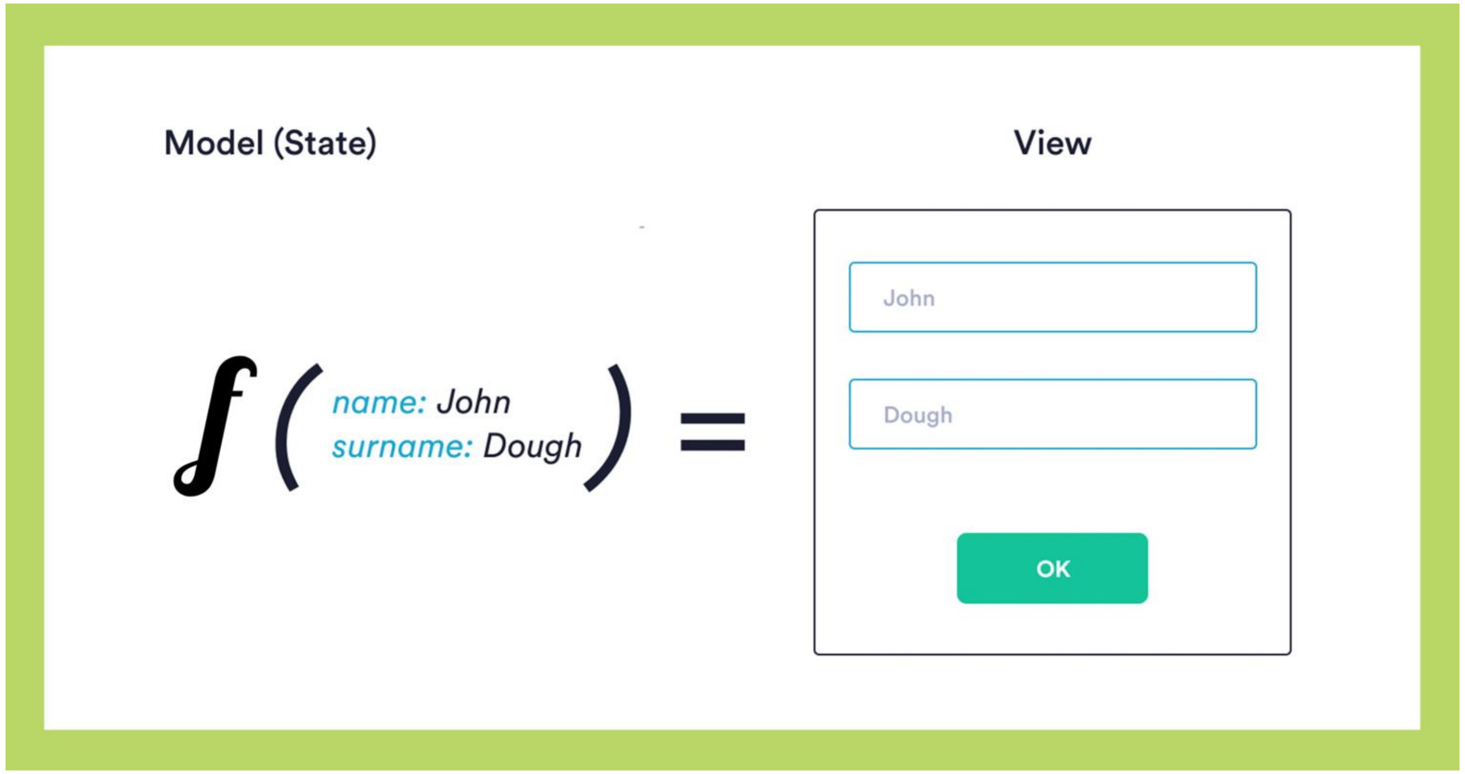
This type of programming is based on mathematical functions and emphasizes the evaluation of expressions rather than the execution of commands. It includes languages such as Haskell, Lisp, and Scheme.



https://realpython.com/python-functional-programming/

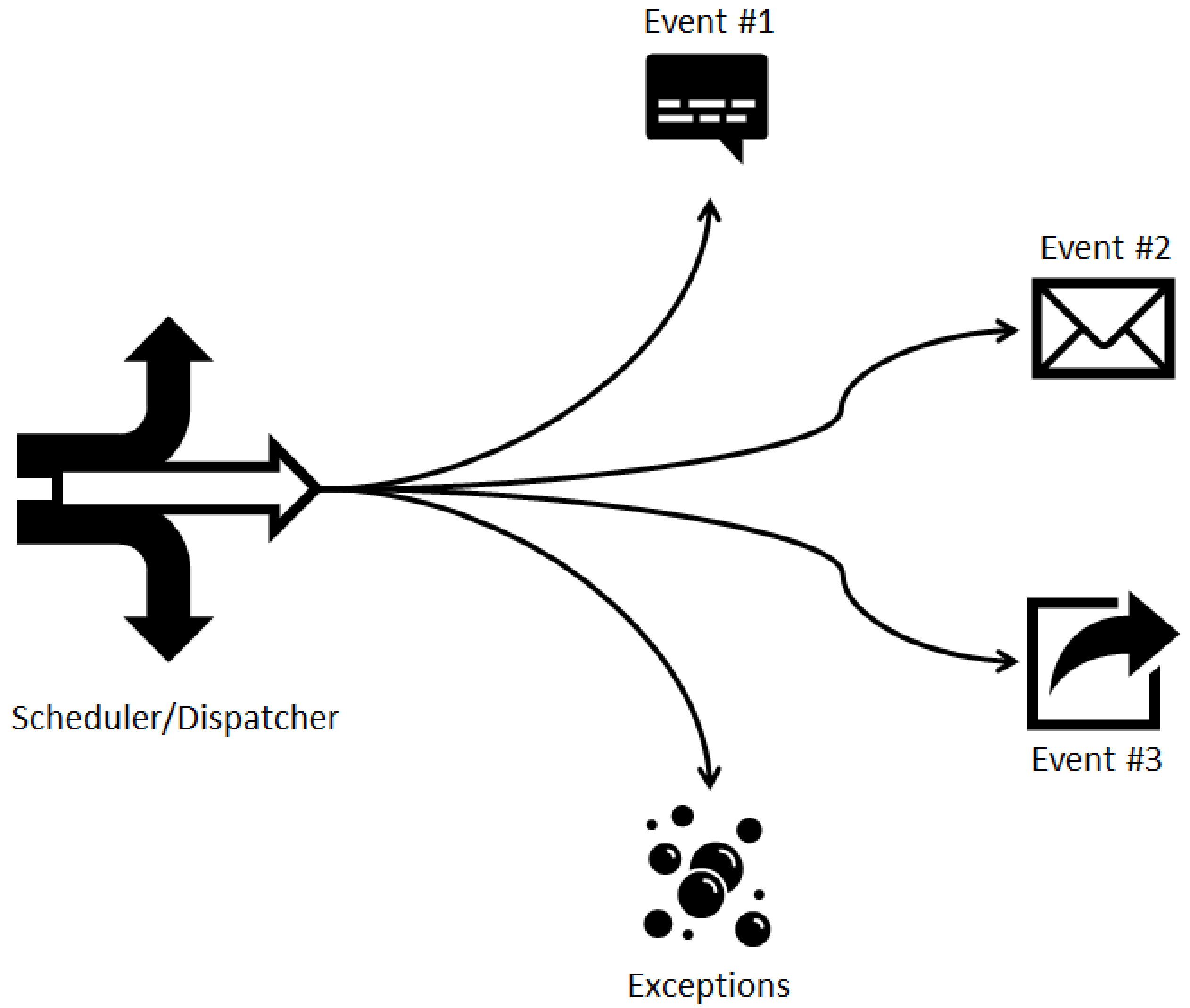
Declarative Programming

This type of programming focuses on describing the desired results rather than specifying the steps to achieve them. It includes languages such as Prolog, SQL, and XSLT.



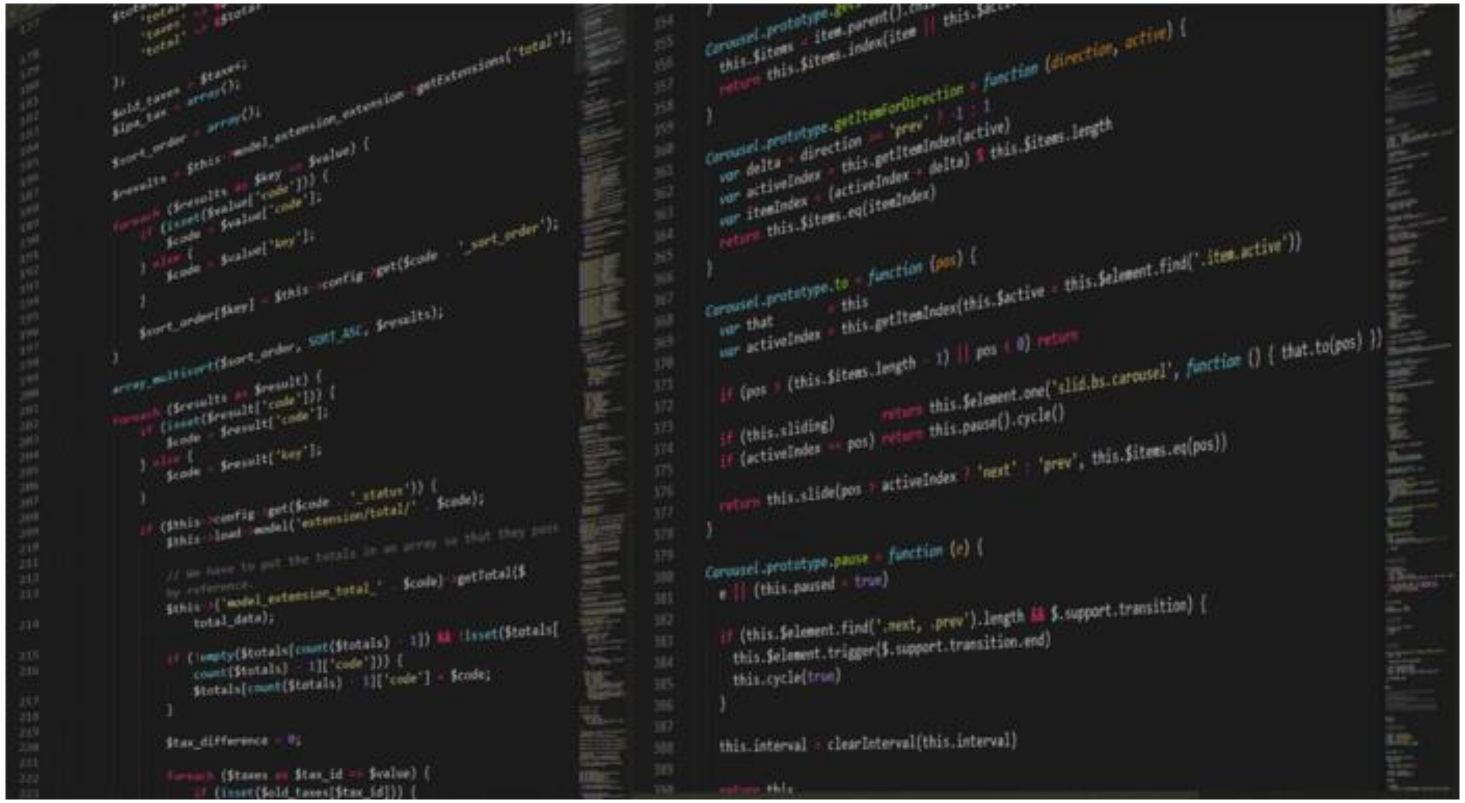
Event-Driven Programming

This type of programming is based on the reaction to events, such as user interactions or incoming data. It includes languages such as JavaScript and Visual Basic for Applications.



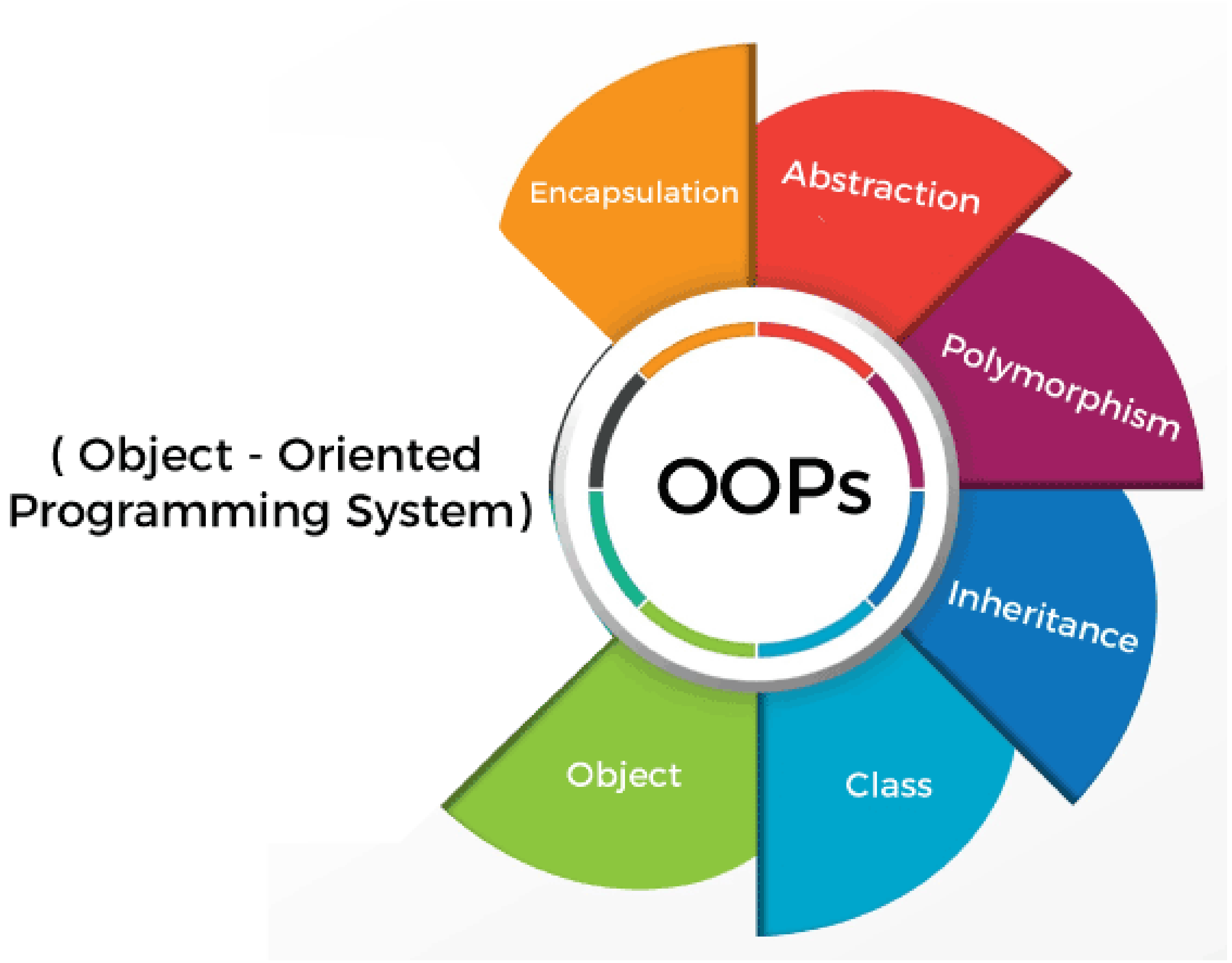
Scripting Programming

This type of programming is typically used for small, quick tasks and automating repetitive procedures. It includes languages such as Perl, PHP, and Python.



Object-Oriented Programming

This type of programming is based on the concept of objects and classes and focuses on encapsulating data and behavior into objects. It includes languages such as Java, Python, and C#.



Outline

Basics of Programming

A set of fundamental concepts that are essential for understanding and writing computer programs.

These concepts form the foundation of computer programming and are used in almost all programming languages.

Understanding these concepts is crucial for writing efficient and effective programs.

Basics of Programming

- Algorithms
- Variables
- Data Types
- Operators
- Conditional Statements
- Loops

Basics of Programming

- **Algorithms**
  - a set of well-defined steps used to solve a problem. Algorithms are the basis of all computer programs.
- **Variables**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.
- **Data Types**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.
- **Operators**
  - Operators are symbols that perform operations on variables, such as addition, subtraction, multiplication, and division.
- **Conditional Statements**
  - Conditional statements allow a program to make decisions based on the values of variables. They include if/else statements, switch statements, and ternary operators.
- **Loops**
  - Loops allow a program to repeat a block of code multiple times, until a certain condition is met. Examples of loops include for loops, while loops, and do/while loops.

Basics of Programming

- **Algorithms**
  - a set of well-defined steps used to solve a problem. Algorithms are the basis of all computer programs.

Basics of Programming

- **Algorithms**
  - a set of well-defined steps used to solve a problem. Algorithms are the basis of all computer programs.
- **Variables**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.

Basics of Programming

- **Algorithms**
  - a set of well-defined steps used to solve a problem. Algorithms are the basis of all computer programs.
- **Variables**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.
- **Data Types**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.

Basics of Programming

- **Algorithms**
  - a set of well-defined steps used to solve a problem. Algorithms are the basis of all computer programs.
- **Variables**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.
- **Data Types**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.
- **Operators**
  - Operators are symbols that perform operations on variables, such as addition, subtraction, multiplication, and division.

Basics of Programming

- **Algorithms**
  - a set of well-defined steps used to solve a problem. Algorithms are the basis of all computer programs.
- **Variables**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.
- **Data Types**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.
- **Operators**
  - Operators are symbols that perform operations on variables, such as addition, subtraction, multiplication, and division.
- **Conditional Statements**
  - Conditional statements allow a program to make decisions based on the values of variables. They include if/else statements, switch statements, and ternary operators.

Basics of Programming

- **Algorithms**
  - a set of well-defined steps used to solve a problem. Algorithms are the basis of all computer programs.
- **Variables**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.
- **Data Types**
  - A variable is a named storage location used to hold data in a program. Variables have a data type, such as integer, string, or boolean, which determines the type of data that can be stored in the variable.
- **Operators**
  - Operators are symbols that perform operations on variables, such as addition, subtraction, multiplication, and division.
- **Conditional Statements**
  - Conditional statements allow a program to make decisions based on the values of variables. They include if/else statements, switch statements, and ternary operators.
- **Loops**
  - Loops allow a program to repeat a block of code multiple times, until a certain condition is met. Examples of loops include for loops, while loops, and do/while loops.

Outline

- Module Induction
- Background to module (the why)
- Introduction to Computer Programming
- Basics of programming
- **Natural Language**
- Translators
- Compilers
- Assemblers

Pause for Question

Computer’s Natural Language

- Formal, human-readable languages used to write instructions for computers to execute.

- Most common types are machine language, assembly language, high–level programming languages, markup languages, scripting languages, query languages, and domain-specific languages.

Computer’s Natural Language

- Computer’s Natural Language is **Machine Language** or **Assembly Language**.
- Machine Language are low-level programming language that consists of binary codes (0s and 1s).
  - 0s and 1s directly represent instructions for the computer to perform.
- Assembly Language is also low-level language but uses mnemonics and symbols to represent machine language instructions. Providing a more readable
- Easy and efficient for the computer but difficult for humans to read and write.

Computer’s Natural Language

High-level programming languages: These are programming languages that use a syntax and structure closer to natural language, making it easier for humans to write, read, and understand code. Examples include **Python**, Java, and C++.

Question

If computers only “understand” Machine Language, how then are they able to understand and process high-level languages, such as Python?

Outline

- Module Induction
- Background to module (the why)
- Introduction to Computer Programming
- Basics of programming
- Natural Language
- **Translators**
- Compilers
- Assemblers

Translators

- Translators are software programs that convert code written in one computer language into another language. There are several types of translators, including:
  - ❑**Compilers**: These translators convert high-level programming code into machine language, which can then be executed directly by the computer's central processing unit (CPU).

Translators

- Translators are software programs that convert code written in one computer language into another language. There are several types of translators, including:
  - ❑**Interpreters:** These translators execute code written in a high-level language line by line, converting each line into machine language as it is executed.

Translators

- Translators are software programs that convert code written in one computer language into another language. There are several types of translators, including:
  - ❑**Assemblers**: These translators convert assembly language code into machine language.

Translators

- Translators are software programs that convert code written in one computer language into another language. There are several types of translators, including:
  - ❑**Linkers:** These translators combine object code generated by compilers into a single executable file.

Translators

- Translators are software programs that convert code written in one computer language into another language. There are several types of translators, including:
  - ❑**Loaders:** These translators load executable code into memory, where it can be executed by the CPU.

Translators

- Translators play a crucial role in the software development process, allowing developers to write code in high-level languages that are easier to read, write, and understand, and then convert that code into machine language that can be executed by the computer.

Outline

- Module Induction
- Background to module (the why)
- Introduction to Computer Programming
- Basics of programming
- Natural Language
- Translators
- **Compilers**
- Assemblers

Compilers

- A translator that takes in high-level programming code and converts it into machine language, which can be executed directly by the computer's CPU.

- A compiler takes the source code written in a high-level language and translates it into an executable program that can run on a computer.

- The process of compiling involves several steps, including lexical analysis, parsing, semantic analysis, code generation, and optimization.

Compilers

- Compilers play a crucial role in the software development process by allowing developers to write code in high-level languages that are easier to read, write, and understand, and then convert that code into machine language that can be executed by the computer.

- They also enable cross-platform development, allowing code written on one operating system to be compiled and run on another.

Compilers

- Examples of popular compilers include GCC (GNU Compiler Collection) for C and C++, the Java compiler for Java, and the Microsoft C++ compiler for C++ on Windows.

Python Compilers?

- For Python, the process of converting code into machine language is different than with traditional compiled languages like C or C++.

- Python is an interpreted language, which means that its code is executed line-by-line by an interpreter, rather than being compiled into machine language beforehand.

- The Python interpreter reads the source code and executes the instructions directly, allowing developers to write, test, and debug their code more quickly and easily than with compiled languages.

- This also makes Python a great language for rapid prototyping and experimentation.

Python Trade-off

- The trade-off for this convenience is that interpreted code is generally slower than compiled code.

- Since the interpreter must translate the code into machine instructions each time it is executed.

- Some tools and techniques, such as pre-compiling the code into bytecode or using just-in-time (JIT) compilation, have been developed to improve the performance of interpreted Python code, but it still remains slower than compiled languages like C or C++.

Hybrid Translators

- Hybrid translators are a type of translator that combine aspects of both compiled and interpreted languages.
- They provide the benefits of both models by translating the code into a more efficient intermediate representation, which is then executed by the computer.
- One example of a hybrid translator is a Just-In-Time (JIT) compiler. A JIT compiler compiles the code on-the-fly, as the program is executed, rather than compiling the entire program beforehand.
- The JIT compiler can optimize the code for the specific environment in which it is running, taking into account the available resources and system architecture.

Hybrid Translators

- Another example of a hybrid translator is a pre-compiler, which converts the code into an intermediate representation that is faster to execute than the original source code.

- The pre-compiler can be used to speed up the startup time of an interpreted language by pre-compiling parts of the code that take a long time to execute.

- An example of a language that uses pre-compilation is Python. In Python, code can be pre-compiled into a binary format called "bytecode". The bytecode is stored in **.pyc** files, which can then be executed by the Python interpreter much faster than the original source code.

Hybrid Translators

- By combining the advantages of both compiled and interpreted languages, hybrid translators provide a way to achieve the best of both worlds, with improved performance and ease of development.

Outline

- Module Induction
- Background to module (the why)
- Introduction to Computer Programming
- Basics of programming
- Natural Language
- Translators
- Compilers
- **Assemblers**

Assembler

- An assembler is a type of computer program that takes assembly language code as input and generates machine code as output.

- Assembly language is a low-level programming language that is used to write instructions for a computer's central processing unit (CPU).

- Assembly language code is more closely related to the machine code executed by the computer than high-level programming languages like Python or Java.

Assembler

- Each instruction in assembly language corresponds directly to a single machine instruction, making it possible to write very efficient and tightly optimized code.
- However, assembly language code is often difficult to read and write, and can be prone to bugs and errors.

- The role of the assembler is to translate the assembly language code into machine code that can be executed by the computer.

- The assembler performs this translation by converting each assembly language instruction into the corresponding machine code instruction and resolving symbolic addresses into actual memory addresses.

Assembler

- The use of assembly language is typically limited to system programming, embedded systems, and other specialized applications where the performance benefits of low-level programming are necessary.

- For most programming tasks, higher-level programming languages like Python, Java, or C are used instead, as they provide a more convenient and readable programming experience, while still offering good performance.

Thank you