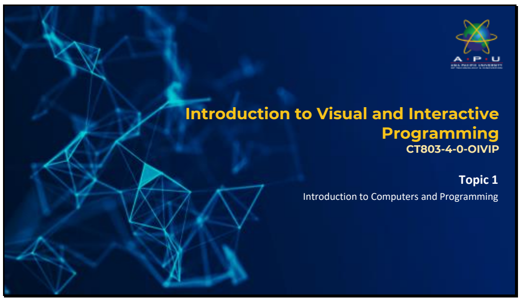
Week 2: Introduction to Computers and Programming

Lecture slides	Your notes here

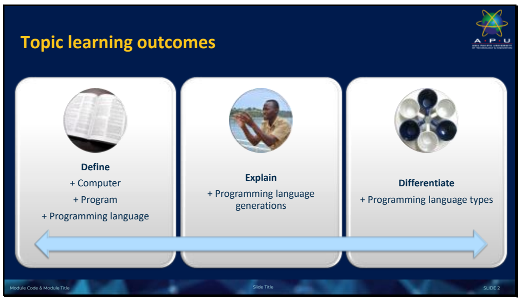


Introduction to Visual and Interactive Programming

CT803-4-0-OIVIP

Topic 1

Introduction to Computers and Programming

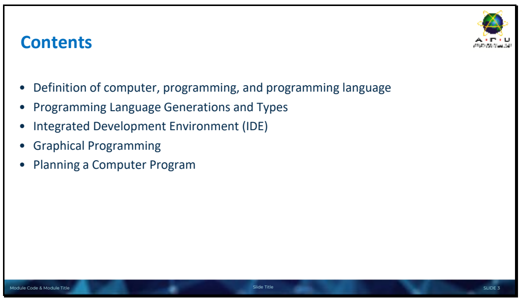


Topic learning outcomes

Define
+ Computer
+ Program
+ Programming language

Explain
+ Programming language generations

Differentiate
+ Programming language types



Contents

- Definition of computer, programming, and programming language
- Programming Language Generations and Types
- Integrated Development Environment (IDE)
- Graphical Programming
- Planning a Computer Program

What is a computer?

A Computer, defined

- Machine that processes information and perform tasks.
- Used in almost every field of human endeavor
  - Science
  - Engineering
  - Business
  - The arts

Computer

- A computer is an **electronic device**, operating under the control of instructions stored in its own memory
  - Accepts **data (input)**,
  - **processes the data** according to specified rules,
  - **produces information (output)**, and
  - **stores the information** for future use
- Most people are inclined to see the computer as smart
- Computers do exactly as humans tell it to do
  - Fast, precise

Computer Program

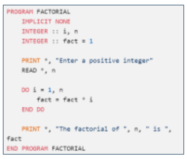
- Tells the computer what to do
- Set of instructions that details ordered operations
  - Algorithms written in programming language and translated to run on a machine
- Perform a specific function(s) or/and achieve specific result(s)
- Program, app and software are sometimes used interchangeably
  - Writing a recipe is writing exact steps to cook something: program
  - A specific and complete recipe can be used to prepare a specific dish: app
  - A complete meal prepared with specific recipe(s) using specific kitchen ware: software

Programming Language

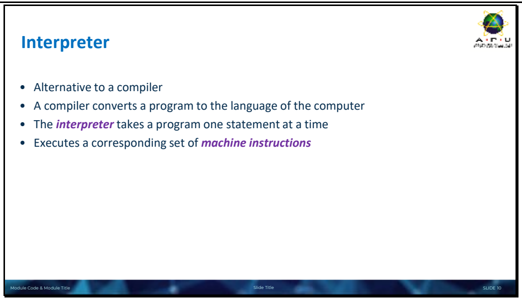
- Set of statements and syntax rules
  - Imagine talking to someone using limited vocabulary and very strict grammar
- Implements sequential, conditional and iterative algorithms
- Programming is also called coding
  - Writing the instruction that the computer will execute
  - High-level programming language allows for English-like sentences
- The program will eventually be turned into machine language
  - Compiler
  - Interpreter

Compiler

- A computer is a machine
  - Uses machine language
  - Executes a program and produces output
- Human programmers write code in human-like language
- A ***compiler*** is a program that converts a program written in a programming language into ***machine language***

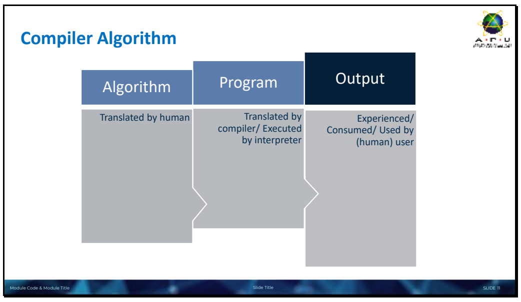


```
PROGRAM FACTORIAL
    IMPLICIT NONE
    INTEGER :: i, n
    INTEGER :: fact = 1

    PRINT *, "Enter a positive integer"
    READ *, n

    DO i = 1, n
        fact = fact * i
    END DO

    PRINT *, "The factorial of ", n, " is ",
fact
END PROGRAM FACTORIAL
```



Interpreter

- Alternative to a compiler
- A compiler converts a program to the language of the computer
- The ***interpreter*** takes a program one statement at a time
- Executes a corresponding set of ***machine instructions***



Compiler Algorithm

Algorithm

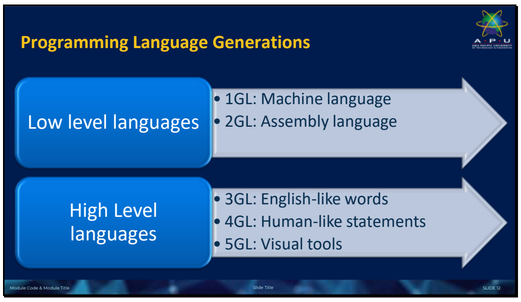
Translated by human

Program

Translated by compiler/ Executed by interpreter

Output

Experienced/ Consumed/ Used by (human) user



Programming Language Generations

Low level languages

- 1GL: Machine language
- 2GL: Assembly language

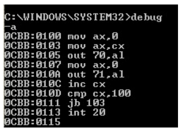
High Level languages

- 3GL: English-like words
- 4GL: Human-like statements
- 5GL: Visual tools

Machine Language

- Binary codes (Base-2)
- Sequence of 1s and 0s that means something
- Simple to represent

Assembly Language

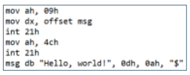


```
C:\WINDOWS\SYSTEM32>debug
-a
0CBB:0100 mov ax,0
0CBB:0103 mov ax,cx
0CBB:0105 out 70,al
0CBB:0107 mov ax,0
0CBB:010A out 71,al
0CBB:010C inc cx
0CBB:010D cmp cx,100
0CBB:0111 jb 103
0CBB:0113 int 20
0CBB:0115
```

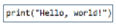
- Abbreviations to represent a command
  - Words and symbols
- Easier to understand compared to machine language
  - Still machine-specific
- Very different from human language

High Level Languages

- English-like environment
  - Easier to remember and figure out commands
- Portable codes
  - Similar codes can run on different devices



```
mov ah, 09h
mov dx, offset msg
int 21h
mov ah, 4ch
int 21h
msg db "Hello, world!", 0dh, 0ah, "$"
```



```
print("Hello, world!")
```

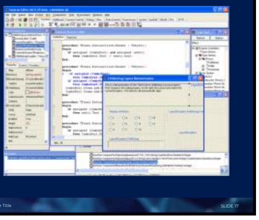
- Examples of 3rd Generation Languages: C, C++ and Java
- Examples of 4th Generation Languages: Perl, Python and Ruby
- Examples of 5th Generation Languages: Mercury, OPS5, and Prolog

Comparison

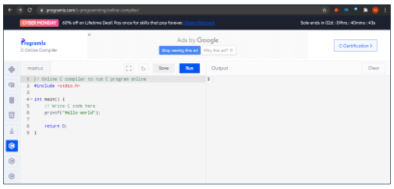
	Machine Language	Assembly Language	High-level Languages
Time to execute	Since it is the basic language of the computer, it does not require any translation, and hence ensures better machine efficiency. This means the programs run faster.	A program called an 'assembler' is required to convert the program into machine language. Thus, it takes longer to execute than a machine language program.	A program called a compiler or interpreter is required to convert the program into machine language. Thus, it takes more time for a computer to execute.
Time to develop	Needs a lot of skill, as instructions are very lengthy and complex. Thus, it takes more time to program.	Simpler to use than machine language, though instruction codes must be memorized. It takes less time to develop programs as compared to machine language.	Easiest to use. Takes less time to develop programs and, hence, ensures better program efficiency.

Integrated Development Environment (IDE)

- Programming is an activity that requires different assets and facilities
  - Code editor
  - Compiler
  - Debugger
- An IDE integrates these assets and facilities in one interface
- Cooking in a kitchen



Integrated Development Environment (IDE)

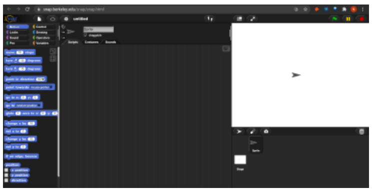

programiz.com/c-programming/online-compiler/
CYBER MONDAY
60% off on Lifetime Deal! Pay once for skills that pay forever. Claim Discount
Sale ends in 02d : 09hrs : 40mins : 43s
Programiz
C Online Compiler
Ads by Google
Stop seeing this ad
Why this ad?
C Certification
main.c
Save
Run
Output
Clear

```
// Online C compiler to run C program online
#include <stdio.h>

int main() {
    // Write C code here
    printf("Hello world");

    return 0;
}
```

```
$
```

Simplified online version of an IDE

Integrated Development Environment (IDE)



Seems familiar?

Graphical Programming

- What we'll learn: Creating an interactive application.
- Basic concepts involved in creating an interactive application:
  - Control of flow
  - Interactive Animations
  - Events
  - Event Handling Methods

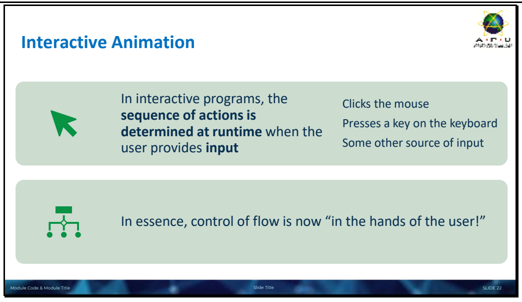
Control of Flow

Control of flow

- How the sequence of actions in a program is controlled
- What action happens first, what happens next, and so on

In movie-style programs the **sequence of actions is determined by the programmer**

- Has a storyline
- Designed sequence planned out by writing program methods

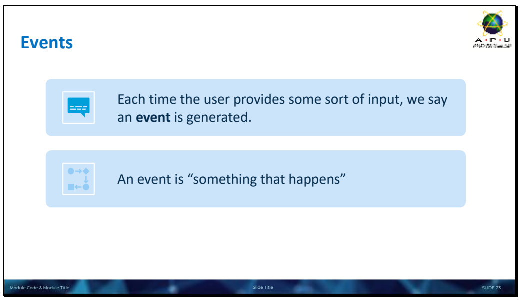


Interactive Animation

In interactive programs, the **sequence of actions is determined at runtime** when the user provides **input**

- Clicks the mouse
- Presses a key on the keyboard
- Some other source of input

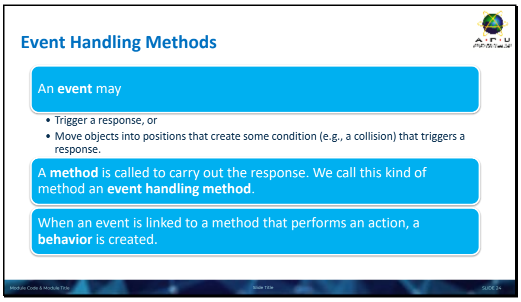
In essence, control of flow is now “in the hands of the user!”



Events

Each time the user provides some sort of input, we say an **event** is generated.

An event is “something that happens”



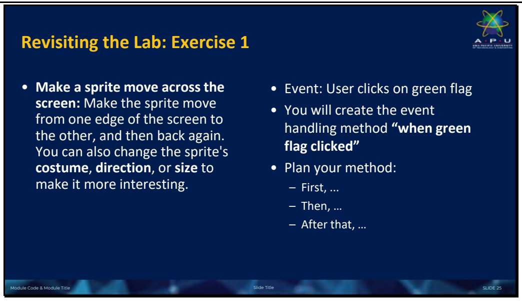
Event Handling Methods

An **event** may

- Trigger a response, or
- Move objects into positions that create some condition (e.g., a collision) that triggers a response.

A **method** is called to carry out the response. We call this kind of method an **event handling method**.

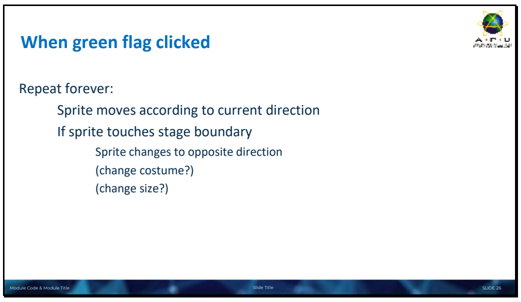
When an event is linked to a method that performs an action, a **behavior** is created.



Revisiting the Lab: Exercise 1

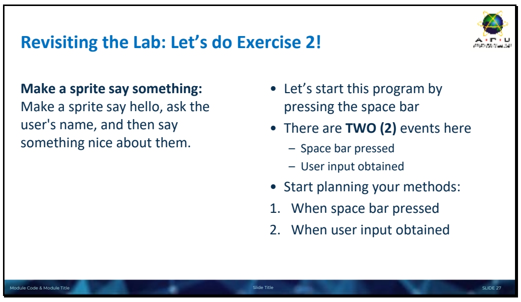
- **Make a sprite move across the screen:** Make the sprite move from one edge of the screen to the other, and then back again. You can also change the sprite's **costume**, **direction**, or **size** to make it more interesting.

- Event: User clicks on green flag
- You will create the event handling method **"when green flag clicked"**
- Plan your method:
  - First, …
  - Then, …
  - After that, …



When green flag clicked

```
Repeat forever:
        Sprite moves according to current direction
        If sprite touches stage boundary
                Sprite changes to opposite direction
                (change costume?)
                (change size?)
```

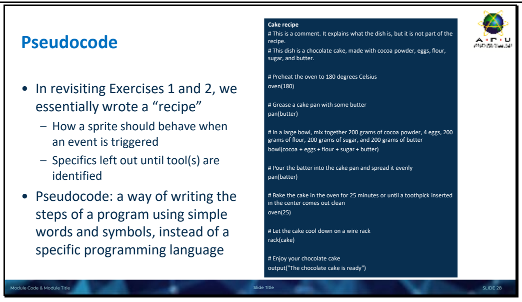


Revisiting the Lab: Let's do Exercise 2!

Make a sprite say something: Make a sprite say hello, ask the user's name, and then say something nice about them.

- Let's start this program by pressing the space bar
- There are **TWO (2)** events here
  - Space bar pressed
  - User input obtained
- Start planning your methods:

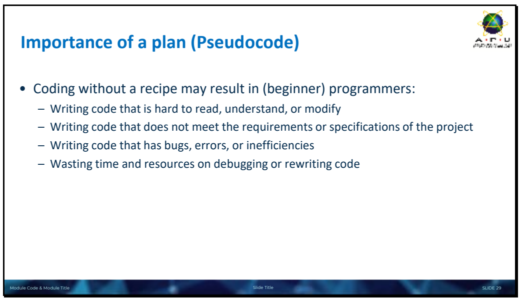
1. When space bar pressed
2. When user input obtained



Pseudocode

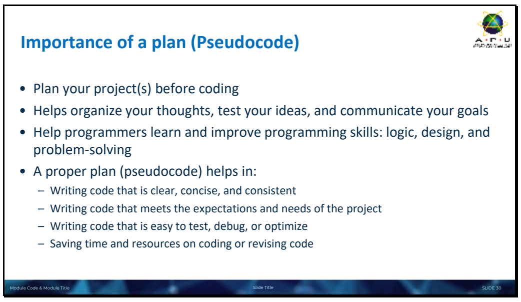
- In revisiting Exercises 1 and 2, we essentially wrote a “recipe”
  - How a sprite should behave when an event is triggered
  - Specifics left out until tool(s) are identified
- Pseudocode: a way of writing the steps of a program using simple words and symbols, instead of a specific programming language

```
Cake recipe
# This is a comment. It explains what the dish is, but it is not part of the recipe.
# This dish is a chocolate cake, made with cocoa powder, eggs, flour, sugar, and butter.

# Preheat the oven to 180 degrees Celsius
oven(180)

# Grease a cake pan with some butter
pan(butter)

# In a large bowl, mix together 200 grams of cocoa powder, 4 eggs, 200 grams of flour, 200 grams of sugar, and 200 grams of butter
bowl(cocoa + eggs + flour + sugar + butter)

# Pour the batter into the cake pan and spread it evenly
pan(batter)

# Bake the cake in the oven for 25 minutes or until a toothpick inserted in the center comes out clean
oven(25)

# Let the cake cool down on a wire rack
rack(cake)

# Enjoy your chocolate cake
output("The chocolate cake is ready")
```



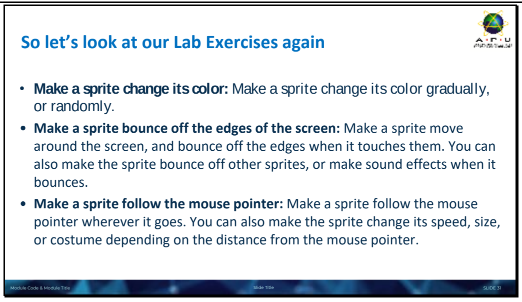
Importance of a plan (Pseudocode)

- Coding without a recipe may result in (beginner) programmers:
  - Writing code that is hard to read, understand, or modify
  - Writing code that does not meet the requirements or specifications of the project
  - Writing code that has bugs, errors, or inefficiencies
  - Wasting time and resources on debugging or rewriting code



Importance of a plan (Pseudocode)

- Plan your project(s) before coding
- Helps organize your thoughts, test your ideas, and communicate your goals
- Help programmers learn and improve programming skills: logic, design, and problem-solving
- A proper plan (pseudocode) helps in:
  - Writing code that is clear, concise, and consistent
  - Writing code that meets the expectations and needs of the project
  - Writing code that is easy to test, debug, or optimize
  - Saving time and resources on coding or revising code



So let’s look at our Lab Exercises again

- **Make a sprite change its color:** Make a sprite change its color gradually, or randomly.
- **Make a sprite bounce off the edges of the screen:** Make a sprite move around the screen, and bounce off the edges when it touches them. You can also make the sprite bounce off other sprites, or make sound effects when it bounces.
- **Make a sprite follow the mouse pointer:** Make a sprite follow the mouse pointer wherever it goes. You can also make the sprite change its speed, size, or costume depending on the distance from the mouse pointer.