**Chapter 1**

**Self-Review Exercises 1.1**

**Fill in the blanks in each of the following statements:**

a) Computers process data under the control of sets of instructions called \_PROGRAMS\_\_\_\_\_\_\_\_\_\_ .

b) The key logical units of the computer are the \_INPUT\_\_,\_\_OUTPUT\_\_,\_MEMORY \_\_\_ ,\_\_ARITHMETIC ,\_LOGIC\_\_ and \_\_CONTROL UNITS\_\_\_\_.

c) The three types of languages they are\_ HIGH LEVEL\_\_\_\_ , \_\_\_ASSEMBLY\_\_\_\_\_\_and \_\_MACHINE LANGUAGES\_\_\_\_\_\_.

d) The programs that translate high-level language programs into machine language are called \_TRANSLATORS\_\_ .

e) \_\_ANDROID\_\_is an operating system for mobile devices based on the Linux kernel and Java.

f) \_STABLE RELEASE\_software is generally feature complete, (supposedly) bug free and ready for use by the community.

g) The Wii Remote, as well as many smartphones, use a(n) \_ACCELEROMETER\_\_which allows the device to respond to motion.

**1.2 Fill in the blanks in each of the following sentences about the Java environment:**

a) The \_JAVA\_ command from the JDK executes a Java application.

b) The \_\_JAVAC\_\_\_\_\_ command from the JDK compiles a Java program.

c) A Java source code file must end with the \_.JAVA\_\_\_\_\_\_ file extension.

d) When a Java program is compiled, the file produced by the compiler ends with the \_.CLASS\_\_\_\_\_\_\_ file extension.

e) The file produced by the Java compiler contains \_\_BYTECODE\_\_\_\_\_\_\_ that are executed by the Java Virtual Machine.

**1.3 Fill in the blanks in each of the following statements**

a) Objects enable the design practice of\_ENCAPSULATION\_\_\_\_\_\_ —although they may know how to communicate with one another across well-defined interfaces, they normally are not allowed to know how other objects are implemented.

b) Java programmers concentrate on creating \_CLASSES\_\_\_\_ , which contain fields and the set of methods that manipulate those fields and provide services to clients.

c) The process of analyzing and designing a system from an object-oriented point of view is called \_OBJECT-ORIENTED PROGRAMMING\_\_\_\_\_\_\_.

d) A new class of objects can be created conveniently by \_INHERITANCE\_\_\_\_\_\_\_\_ —the new class (called the subclass) starts with the characteristics of an existing class (called the superclass), possibly customizing them and adding unique characteristics of its own.

e) \_\_\_UNIFIED MODELLING LANGUAGE\_\_\_\_\_ is a graphical language that allows people who design software systems to use an industry-standard notation to represent them.

f) The size, shape, color and weight of an object are considered\_ATTRIBUTES\_\_\_\_\_\_ of the object’s class.

**Exercises 1.4 Fill in the blanks in each of the following statements:**

a) The logical unit that receives information from outside the computer for use by the computer is the \_INPUT UNIT\_\_\_\_\_\_ .

b) The process of instructing the computer to solve a problem is called \_PROGRAMMING\_\_\_\_ .

c) \_HIGH LEVEL LANGUAGE\_\_\_\_\_ is a type of computer language that uses English-like abbreviations for machine-language instructions.

d) \_OUTPUT UNIT\_\_\_\_\_\_\_ is a logical unit that sends information which has already been processed by the computer to various devices so that it may be used outside the computer.

e) \_MEMORY UNIT\_\_\_and \_CONTROL UNIT\_\_\_\_\_ are logical units of the computer that retain information.

f) ARITHMETIC UNIT\_\_\_\_ is a logical unit of the computer that performs calculations. g) \_LOGIC UNIT\_\_\_\_\_\_\_ is a logical unit of the computer that makes logical decisions.

h) \_HIGH LEVEL\_\_\_\_\_\_\_\_\_ languages are most convenient to the programmer for writing programs quickly and easily.

i) The only language a computer can directly understand is that computer’s\_MACHINE CODE\_\_\_\_\_\_\_\_.

j) \_CONTROL UNIT\_\_\_\_\_\_\_\_ is a logical unit of the computer that coordinates the activities of all the other logical units.

**1.5 Fill in the blanks in each of the following statements:**

a) The \_\_JAVA\_\_\_\_\_\_\_ programming language is now used to develop large-scale enterprise applications, to enhance the functionality of web servers, to provide applications for consumer devices and for many other purposes. b) \_\_C\_\_\_\_\_\_\_ initially became widely known as the development language of the UNIX operating system.

c) The \_TRANSMISSION CONTROL PROTOCOL\_\_ ensures that messages, consisting of sequentially numbered pieces called bytes, were properly routed from sender to receiver, arrived intact and were assembled in the correct order.

d) The \_\_\_C++\_\_\_\_\_\_\_\_\_ programming language was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories.

1.6 Fill in the blanks in each of the following statements:

a) Java programs normally go through five phases— \_EDIT\_\_\_\_\_, \_COMPILE\_\_\_\_\_\_ ,\_\_LOAD\_\_\_\_\_\_\_ , \_\_VERIFY\_\_\_\_\_\_\_ and \_EXECUTE\_\_\_\_\_\_\_\_ .

b) A(n)\_\_INTEGRATED DEVELOPMENT ENVIRONMENT\_\_\_\_\_\_\_\_ provides many tools that support the software development process, such as editors for writing and editing programs, debuggers for locating logic errors in programs, and many other features.

c) The command java invokes the \_JAVA VIRTUAL MACHINE\_\_\_\_\_\_\_\_\_, which executes Java programs.

d) A(n) \_VIRTUAL MACHINE\_\_\_\_\_\_\_ is a software application that simulates a computer, but hides the underlying operating system and hardware from the programs that interact with it.

e) The \_CLASS LOADER\_\_\_ takes the .class files containing the program’s bytecodes and transfers them to primary memory. f) The examines bytecodes to ensure that they’re valid.

f) The \_BYTECODE VERIFIER\_\_\_\_\_\_\_\_\_ examines bytecodes to ensure that they’re valid.

1.7 Explain the two compilation phases of Java programs.

**PHASE 1 — Compilation (Source Code → Bytecode)**

**🧩 What Happens:**

1. You write your program in a .java file using human-readable Java code.
2. public class Hello {
3. public static void main(String[] args) {
4. System.out.println("Hello, Java!");
5. }
6. }
7. You compile it using the **Java compiler (javac)**:
8. javac Hello.java
9. The **Java compiler checks**:
   * Syntax errors
   * Type correctness
   * Class definitions
   * Method declarations, etc.
10. If everything is correct, it **does not generate machine code yet**.  
    Instead, it produces an **intermediate file** called **bytecode** (saved as Hello.class).

**Output of Phase 1:**

👉 **Bytecode (.class file)**

Bytecode is:

* Platform-independent
* Not tied to any specific computer hardware
* An intermediate representation of your program

Think of it as:

“A universal instruction set that any computer can understand *if it has a Java Virtual Machine (JVM)*.”

**PHASE 2 — Execution (Bytecode → Machine Code)**

**🧠 What Happens:**

1. When you run the program:
2. java Hello
3. The **Java Virtual Machine (JVM)** loads the .class file.
4. Inside the JVM:
   * The **Class Loader** loads classes into memory.
   * The **Bytecode Verifier** checks for illegal code (e.g., security violations).
   * The **Interpreter / JIT Compiler (Just-In-Time)** translates the **bytecode into native machine code** for your computer’s CPU.
5. Finally, the **machine code executes** line by line on your processor, producing the output:
6. Hello, Java!

1.8 One of the world’s most common objects is a wrist watch. Discuss how each of the following terms and concepts applies to the notion of a watch: object, attributes, behaviors, class, inheritance (consider, for example, an alarm clock), modeling, messages, encapsulation, interface and information hiding.

**1. Object**

An **object** is a specific instance of something that has characteristics (attributes) and can perform actions (behaviors).

* In this case, **your wristwatch** — the one on your hand — is an object.
* It exists in the real world and can perform tasks like showing the time, date, or even heart rate (for smartwatches).

**2. Attributes**

**Attributes** (also called *properties* or *fields*) describe the **state** or **features** of an object.  
For a wristwatch, attributes might include:

* brand (e.g., Rolex, Casio, Apple)
* color (e.g., black, silver)
* type (analog or digital)
* time (current time displayed)
* batteryLevel
* strapMaterial

These are the **data members** that define what makes each watch unique.

**3. Behaviors**

**Behaviors** (or *methods*) describe what the object **can do** or **what can be done to it**.  
A wristwatch’s behaviors might include:

* showTime()
* setTime()
* showDate()
* startStopwatch()
* setAlarm()

These define the **actions** the object can perform — just like functions in code.

**4. Class**

A **class** is a **blueprint** or **template** used to create objects.  
Think of a **"Watch" class** as the general model that defines what all watches should have:

* **Attributes:** brand, color, type, time
* **Behaviors:** showTime(), setTime(), showDate()

Every specific watch (Rolex, Seiko, Apple Watch) is then an **object (instance)** of that **Watch class**.

**5. Inheritance**

**Inheritance** allows one class to **reuse** and **extend** another class’s properties and behaviors.  
For example:

* A **base class** could be Watch.
* A **subclass** could be AlarmClock or SmartWatch.

The **AlarmClock** inherits everything a Watch has (time, display, setTime()), but also adds new functionality like setAlarm() or ringAlarm().  
This avoids rewriting code and models real-world relationships.

**6. Modeling**

**Modeling** is the process of **representing real-world entities** (like a watch) in a structured, programmable form.  
When you model a watch in software, you define its **attributes and behaviors** so a computer can simulate how it works.  
Example model (simplified in pseudocode):

class Watch {

String brand;

String color;

void showTime() { ... }

void setTime() { ... }

}

Modeling helps bridge the gap between **real-world understanding** and **computer representation**.

**7. Messages**

In OOP, **messages** are how objects **communicate** with each other — usually through **method calls**.  
For example:

* When your phone sends a signal to your smartwatch to sync time, it’s sending a **message** (a request) like updateTime().
* The watch receives the message and executes the method.

Messages = communication between objects.

**8. Encapsulation**

**Encapsulation** means **bundling** data (attributes) and methods (behaviors) inside a single unit — the object — and **restricting direct access** to the inner workings.  
For instance, you can’t directly change the time gears inside your digital watch — you press buttons or use an app (the public interface).  
In code:

class Watch {

private String time;

public void setTime(String newTime) { time = newTime; }

public String getTime() { return time; }

}

The time variable is **protected**; access is only allowed through public methods.  
This ensures **data integrity**.

**9. Interface**

An **interface** defines **what actions** an object can perform, **without specifying how** they’re done.  
For example, a watch’s interface to the user is its **buttons** or **touchscreen** — you can press a button to set the alarm, but you don’t need to know the internal circuitry.  
In code, an interface might look like this:

interface TimeDisplay {

void showTime();

void setTime();

}

Different classes (analog watch, digital watch, smartwatch) can **implement** this interface in their own way.

**10. Information Hiding**

**Information hiding** is closely related to encapsulation — it means concealing **implementation details** that users or other objects don’t need to know.  
You don’t know how your watch internally counts seconds — you just see the hands move or digits change.  
In programming, this keeps the system **modular and secure**, allowing changes inside a class without affecting the outside code.

**Making a Difference**

***1.9 (Test-Drive: Carbon Footprint Calculator)***

Some scientists believe that carbon emissions, especially from the burning of fossil fuels, contribute significantly to global warming and that this can be combatted if individuals take steps to limit their use of carbon-based fuels. Organizations and individuals are increasingly concerned about their “carbon footprints.” Websites such as TerraPass http://www.terrapass.com/carbon-footprint-calculator/ and Carbon Footprint http://www.carbonfootprint.com/calculator.aspx provide carbon-footprint calculators. Test-drive these calculators to determine your carbon footprint. Exercises in later chapters will ask you to program your own carbon-footprint calculator. To prepare for this, use the web to research the formulas for calculating carbon footprints.

### TerraPass (terrapass.com)

* TerraPass offers individual, business, and event calculators, letting users input data on **home energy use**, **ground travel**, **flights / air travel**, and **waste**. [Terrapass+5Terrapass+5Terrapass+5](https://terrapass.com/carbon-footprint-calculator/?utm_source=chatgpt.com)
* For the **flights** component, they ask for origin/destination, compute a distance (great circle or some routing), and then apply emission factors (kg CO₂e per passenger-km) for different flight lengths. [Terrapass+2Terrapass+2](https://terrapass.com/flight-carbon-calculator/?utm_source=chatgpt.com)
* For **shipping or freight** they use emissions = weight × distance × factor (depending on transport mode) to estimate CO₂ emissions. [Terrapass](https://terrapass.com/shipping-carbon-footprint-calculator/?utm_source=chatgpt.com)
* Their methodology generally relies on standard “emission factors” (e.g. kg CO₂ per kWh of electricity, or per liter of gasoline) multiplied by your usage. [Terrapass+4Mapbox+4Terrapass+4](https://www.mapbox.com/blog/how-we-went-carbon-neutral?utm_source=chatgpt.com)
* They also reference the GHG Protocol (especially for business emission accounting) in their business offering. [Terrapass](https://terrapass.com/product/business-carbon-footprinting/?utm_source=chatgpt.com)
* One independent review notes that TerraPass uses U.S. EPA and GHG Protocol data, but their scope (especially Scope 3 emissions) is more limited. [climatiq.io](https://www.climatiq.io/blog/list-free-business-carbon-footprint-calculators?utm_source=chatgpt.com)

So, TerraPass is a “practical, user-friendly” calculator: it collects a few key inputs (energy, travel, waste) and applies well-established emission factors to produce a CO₂e (carbon dioxide equivalent) footprint.

### CarbonFootprint.com (carbonfootprint.com)

* Their calculator lets you estimate emissions from **buildings**, **cars / ground transport**, **flights**, and “other” activities. [carbonfootprint.com](https://www.carbonfootprint.com/calculator.aspx?utm_source=chatgpt.com)
* They have a “classic” (older) and new “Sustrax” calculators, but the underlying principle is the same: convert usage (kWh, liters, miles, etc.) → CO₂e. [carbonfootprint.com](https://www.carbonfootprint.com/calculator.aspx?utm_source=chatgpt.com)
* Their calculators are backed by two decades of experience in emissions estimation. [carbonfootprint.com](https://www.carbonfootprint.com/calculator.aspx?utm_source=chatgpt.com)

So CarbonFootprint.com is somewhat more global / general, with support for more categories, but uses essentially the same principles.

When using either tool, you’ll typically:

1. Enter your **yearly / monthly usage** data (electricity, fuel, miles traveled, flights, waste)
2. The tool computes emissions for each category using emission factors
3. It **sums** all those emissions to give your total carbon footprint (usually in metric tons CO₂e)
4. It may allow you to “offset” (“neutralize”) some emissions by purchasing carbon credits, etc.

If you try the calculators with your own data (electricity bill, miles driven, flights taken), you’ll get a number (e.g. “X metric tons of CO₂e per year”). That’s your personal carbon footprint (as estimated with their assumptions).

## 2. The formulas & methodology behind carbon footprint calculation

Now let’s dig into the formulas and principles you’d use when you write your **own** carbon-footprint calculator.

### Core idea: Emissions = Activity × Emission Factor

The central formula is:

Emission (kg CO₂e) = Activity Amount × Emission Factor

* **Activity Amount** is how much “thing” you used (e.g. kWh of electricity, liters of petrol, miles driven, flight-kilometers)
* **Emission Factor** is how much CO₂ (or GHG equivalent) is emitted per unit of that “thing” (e.g. kg CO₂ per kWh, kg CO₂ per liter petrol, kg CO₂ per passenger-kilometer flown)

You then sum up emissions from all your emission sources to get your total footprint.

A more refined formula (allowing for reductions) is:

E = A × EF × (1 − ER/100)

* **E** = emissions
* **A** = activity rate (how much energy / distance / usage)
* **EF** = emission factor
* **ER** = efficiency / reduction percentage (if you have mitigation) [diginex.com](https://www.diginex.com/insights/how-to-calculate-carbon-emissions?utm_source=chatgpt.com)

Another variant, used especially in business / industrial settings:

Emissions = ∑ (Consumption\_i × EmissionFactor\_i)

Sum over i (each energy/fuel/transport category) [carbonfootprint.com+4Business Energy Scotland+4Brightly+4](https://businessenergyscotland.org/guides/how-calculate-your-businesss-carbon-footprint/?utm_source=chatgpt.com)

### Emission factors, units, and CO₂e

* Emission factors vary by **energy source**, **fuel**, **region**. E.g., 1 kWh of coal-fired electricity produces more CO₂ than 1 kWh from natural gas or solar. [Center for Sustainable Systems+2enelgreenpower.com+2](https://css.umich.edu/publications/factsheets/sustainability-indicators/carbon-footprint-factsheet?utm_source=chatgpt.com)
* Also, there are **multiple greenhouse gases** (CO₂, CH₄, N₂O, etc.). To compare them, we convert them all into **CO₂ equivalents (CO₂e)** using **Global Warming Potential (GWP)** multipliers. For example, methane has a much higher GWP per unit mass than CO₂. [Wikipedia](https://en.wikipedia.org/wiki/Global_warming_potential?utm_source=chatgpt.com)
* The boundary of what you include matters. Some methodologies classify emissions into **Scope 1, 2, 3** (direct, indirect, supply-chain) when doing business emissions. [Brightly+3PwC+3Terrapass+3](https://www.pwc.com/sk/en/environmental-social-and-corporate-governance-esg/measuring-environmental-impact/carbon-footprint-calculation.html?utm_source=chatgpt.com)

### Steps to build your own calculator

Here’s a typical workflow:

1. **Define the scope / boundary**
   * Which emission sources will you include?
     + Home energy (electricity, gas)
     + Transportation (cars, buses, flights)
     + Waste / recycling
     + Others (food, goods, services)
   * Over what time period? (typically yearly)
2. **Collect data**
   * Your electricity usage (kWh per year)
   * Fuel usage (liters of petrol / diesel)
   * Miles driven / kilometers traveled
   * Flights taken (distance)
   * Waste amounts (how much you throw away / recycle)
3. **Obtain emission factors**
   * Find tables / databases (e.g. EPA, IPCC, national sources) giving “kg CO₂e per unit”
   * Use that appropriate to your region / fuel / mode
4. **Compute emissions for each category**
   * Multiply usage by emission factor
5. **Sum all emissions** → total carbon footprint
6. (Optional) **Offset or subtract mitigations**
   * If you plant trees, buy carbon credits, etc., subtract that from your total
7. **Report results (e.g. in metric tons of CO₂e)**

### Example (toy, simplified)

Suppose in one year you tell me:

* Electricity use: 1,200 kWh
* Car travel: 10,000 km with petrol
* One round-trip flight: 2,000 km

We might have emission factors:

* Electricity: 0.5 kg CO₂e per kWh
* Petrol car: 0.2 kg CO₂e per km
* Flight: 0.15 kg CO₂e per passenger-km

Then:

* Home emissions = 1,200 × 0.5 = 600 kg CO₂e
* Car emissions = 10,000 × 0.2 = 2,000 kg CO₂e
* Flight emissions = 2,000 × 0.15 = 300 kg CO₂e

Total = 600 + 2,000 + 300 = **2,900 kg CO₂e** (or 2.9 metric tons CO₂e)

Of course, in a “real” calculator you’d include more sources and more precise factors.

## 3. Pseudocode / structure for your own carbon footprint calculator

Here is a rough sketch in pseudocode:

# Data structure to hold emissions factors (units appropriate)

emission\_factors = {

"electricity\_kWh": 0.5, # kg CO2e per kWh (example)

"petrol\_km": 0.2, # kg CO2e per km by petrol car

"flight\_km": 0.15, # kg CO2e per passenger-km by air

# … other sources

}

def compute\_emission(activity\_type, usage\_amount):

"""Compute emissions for one activity."""

factor = emission\_factors[activity\_type]

return usage\_amount \* factor

def total\_footprint(user\_inputs):

"""user\_inputs is a dict with keys: electricity\_kWh, petrol\_km, flight\_km, etc."""

total = 0.0

for activity, usage in user\_inputs.items():

total += compute\_emission(activity, usage)

return total # in kg CO2e

# Example use:

user\_data = {

"electricity\_kWh": 1200,

"petrol\_km": 10000,

"flight\_km": 2000

}

footprint\_kg = total\_footprint(user\_data)

footprint\_tons = footprint\_kg / 1000.0

print("Your carbon footprint = ", footprint\_tons, "tons CO2e")

You could expand this:

* Allow multiple flights (sum over them)
* Include waste, recycling
* Include emissions from food, goods (using approximate factors)
* Allow offsets (subtracting a mitigation amount)
* Use region-specific emission factor tables

## 4. Tips / caveats & challenges

* **Emission factors vary**: The factor for electricity depends heavily on how electricity is generated (coal, gas, hydro, renewables) in your region.
* **Boundary issues**: Deciding whether to include upstream emissions (manufacturing, transport of goods) is tricky.
* **Indirect vs direct emissions**: Some emissions are not under your direct control (e.g. the emissions of producing the goods you buy).
* **Uncertainty**: Many inputs are estimates; small errors can propagate.
* **Temporal changes**: Emission factors and energy mixes evolve over time — what’s accurate today may be outdated eventually.
* **Comparability**: Different calculators use different assumptions and boundaries, so two tools may give different results even for the same person.

***1.10 (Test-Drive: Body Mass Index Calculator)***

Obesity causes significant increases in illnesses such as diabetes and heart disease. To determine whether a person is overweight or obese, you can use a measure called the body mass index (BMI). The United States Department of Health and Human Services provides a BMI calculator at http://www.nhlbi.nih.gov/guidelines/obesity/BMI/ bmicalc.htm. Use it to calculate your own BMI. A forthcoming exercise will ask you to program your own BMI calculator. To prepare for this, use the web to research the formulas for calculating BMI.

## 1. What is BMI (Body Mass Index) — formula & meaning

The **Body Mass Index (BMI)** is a standardized measure that relates a person’s **weight** to their **height**, with the intent of estimating whether they are underweight, “normal,” overweight, or obese. It does not directly measure body fat, but is widely used because of its simplicity and broad correlation across populations. [CDC+2Wikipedia+2](https://www.cdc.gov/growth-chart-training/hcp/using-bmi/body-mass-index.html?utm_source=chatgpt.com)

### Standard (metric) formula

The most common formula (in metric units) is:

BMI=weight in kilograms(height in meters)2\text{BMI} = \frac{\text{weight in kilograms}}{(\text{height in meters})^2}BMI=(height in meters)2weight in kilograms​

So if your weight is WWW kg and your height is HHH m, then:

BMI=WH2\text{BMI} = \dfrac{W}{H^2}BMI=H2W​

Alternatively, if you know height in **centimeters (cm)**, you can rewrite:

BMI=W(Hcm/100)2=WHcm2×10,000\text{BMI} = \frac{W}{(H\_{\text{cm}}/100)^2} = \frac{W}{H\_{\text{cm}}^2} \times 10,000BMI=(Hcm​/100)2W​=Hcm2​W​×10,000

This is just algebraic rearrangement.CDC+1

### Alternative (U.S. customary) formula

If you measure weight in **pounds (lb)** and height in **inches (in)**, there’s a conversion factor to get the same scale (kg / m²). The formula becomes:

BMI=weight in lb(height in in)2×703\text{BMI} = \frac{\text{weight in lb}}{(\text{height in in})^2} \times 703BMI=(height in in)2weight in lb​×703

That “× 703” is the conversion factor to map (lb / in²) into kg / m² units.

## 2. BMI categories / interpretation

Once you compute your BMI, you compare it against typical ranges:

| **BMI Value** | **Category** |
| --- | --- |
| < 18.5 | Underweight |
| 18.5 – 24.9 | Normal / Healthy weight |
| 25.0 – 29.9 | Overweight |
| ≥ 30.0 | Obesity |

Some further subclassifications exist (Class I, II, III obesity) depending on how high the BMI is.

The U.S. National Heart, Lung, and Blood Institute (NHLBI) also uses those thresholds.

## 3. Practical example (you can try this yourself)

You can use the NHLBI / NIH BMI calculator (via their website) to plug in your weight and height (in pounds & inches, or kilograms & meters) and it will output your BMI and tell you the category.

If you like, you can give me your weight and height (in your preferred units), and I’ll calculate your BMI for you and tell you which category you fall into.

## 4. Pseudocode / structure for a BMI calculator

Here’s how you might design a simple BMI calculator (in pseudocode):

# Input: weight, height, and units

# E.g. user gives (weight, height, unitSystem) where unitSystem ∈ {"metric", "imperial"}

function computeBMI(weight, height, unitSystem):

if unitSystem == "metric":

# weight in kg, height in meters

bmi = weight / (height \* height)

else if unitSystem == "imperial":

# weight in lb, height in inches

bmi = (weight / (height \* height)) \* 703

else:

error("unknown unit system")

return bmi

function classifyBMI(bmi):

if bmi < 18.5:

return "Underweight"

else if 18.5 ≤ bmi < 25.0:

return "Normal / Healthy weight"

else if 25.0 ≤ bmi < 30.0:

return "Overweight"

else:

return "Obese"

# Example usage:

weight = input("Enter weight (kg or lb): ")

height = input("Enter height (meters or inches): ")

units = input("metric or imperial? ")

bmi\_value = computeBMI(weight, height, units)

category = classifyBMI(bmi\_value)

print("Your BMI is:", bmi\_value)

print("That is considered:", category)

You can add validation (ensure height > 0, weight > 0), rounding, and better reporting.

## 5. Limitations and caveats of BMI

While BMI is very useful at the population level, it has known limitations for individuals:

* **Does not distinguish body composition**: Someone with high muscle mass (e.g. athletes) could have a high BMI but low body fat.
* **Does not indicate fat distribution**: Abdominal (visceral) fat is riskier than fat distributed elsewhere, but BMI doesn't tell you where the fat is.
* **Age, gender, and ethnicity effects**: The same BMI may carry different risks in different populations.
* **Bone density & frame size**: People with larger skeletons or heavier bones may appear heavier.
* Because of these, medical professionals often use **waist circumference**, **waist-to-hip ratio**, or more direct measures (e.g. DXA scans) in addition to BMI.

Also, there’s active discussion in modern medical research about whether BMI should **no longer be the sole criterion** for diagnosing obesity. Some experts propose combining BMI with **waist measurements** and diagnosing “clinical obesity” only when health complications (like diabetes or heart disease) are present.

***1.11 (Attributes of Hybrid Vehicles)***

Hybrid vehicles are becoming increasingly popular, because they often get much better mileage than purely gasoline-powered vehicles. Browse the web and study the features of four or five of today’s popular hybrid cars, then list as many of their hybrid-related attributes as you can. Some common attributes include city-miles-per-gallon and highway-miles-per-gallon. Also list the attributes of the batteries (type, weight, etc.).

***1.12 (Gender Neutrality)***

Many people want to eliminate sexism in all forms of communication. You’ve been asked to create a program that can process a paragraph of text and replace gender-specific words with gender-neutral ones. Assuming that you’ve been given a list of gender-specific words and their gender-neutral replacements (e.g., replace both “wife” and “husband” with “spouse,” “man” and “woman” with “person,” “daughter” and “son” with “child”), explain the procedure you’d use to read through a paragraph of text and manually perform these replacements. How might your procedure generate a strange term like “woperchild?” You’ll soon learn that a more formal term for “procedure” is “algorithm,” and that an algorithm specifies the steps to be performed and the order in which to perform them. We’ll show how to develop algorithms then convert them to Java programs which can be run on computers.

We want a **procedure (algorithm)** that reads a paragraph of text and replaces certain **gender-specific words** with **gender-neutral equivalents.**

Example replacement pairs:

| **Gendered words** | **Gender-neutral replacement** |
| --- | --- |
| wife, husband | spouse |
| man, woman | person |
| daughter, son | child |

### 🪜 Step-by-Step Algorithm (Manual or Conceptual Version)

1. **Start**
   * Begin with a paragraph of text, e.g.  
     “The man and his wife have a daughter.”
2. **Input the list of replacement pairs**
   * You might store them like this (conceptually):
   * wife → spouse
   * husband → spouse
   * man → person
   * woman → person
   * daughter → child
   * son → child
3. **Read the paragraph word by word**
   * You can split the text into individual words or tokens (e.g., by spaces and punctuation).
4. **Check each word**
   * For every word, see if it appears in the list of gender-specific words.
5. **Replace if needed**
   * If it matches, replace it with the gender-neutral word.
   * If not, keep it unchanged.
6. **Rebuild the paragraph**
   * After all replacements, join the words back into a complete paragraph.
7. **Display the updated paragraph**
   * Example output:  
     “The person and their spouse have a child.”
8. **End**

### 💡 Example (Step-by-Step Execution)

**Input:**  
"The man and his wife have a daughter."

**Process:**

| **Original Word** | **Found in List?** | **Replacement** |
| --- | --- | --- |
| The | No | The |
| man | Yes | person |
| and | No | and |
| his | No | his |
| wife | Yes | spouse |
| have | No | have |
| a | No | a |
| daughter | Yes | child |

**Output:**  
"The person and his spouse have a child."

### ⚠️ Why Might You Get a Strange Word Like “woperchild”?

That happens if your **replacement procedure doesn’t separate words properly** or **doesn’t handle overlaps carefully**.

For example:

* Suppose you go through the text **one character at a time** instead of one word at a time.
* You might replace “woman” with “person,” but if you later replace “man” with “person,” you could accidentally replace part of a word that’s already been changed.

Let’s simulate:

Original text: "woman and son"

1. Replace “woman” → “woperson”
2. Then replace “man” → “person”
   * The “man” part in “woperson” still matches “man” (last three letters), so now it becomes "woperperson"

Now imagine similar overlapping replacements across “woman” + “daughter” → could yield "woperchild" if both were processed incorrectly.

### 🧩 Why Does This Happen?

Because the algorithm doesn’t:

* Treat **whole words** separately (it treats “man” inside “woman” as a separate word).
* Or doesn’t **preserve spacing and boundaries**.

This is a **string-processing pitfall** known as **substring collision**.

### ✅ How to Prevent It

1. **Work with complete words** (tokenize by spaces/punctuation).  
   Don’t replace text fragments inside larger words.
2. **Use word boundaries** (e.g., \b in regular expressions in programming languages).
3. **Make replacements in one pass** instead of repeatedly modifying the same text.

### 🧩 In Summary — The Algorithm in Pseudocode

START

Read the paragraph into memory

Load the dictionary of replacements (word pairs)

Split the paragraph into words

FOR each word in the paragraph

IF word exists in the dictionary

Replace it with its gender-neutral equivalent

ENDIF

ENDFOR

Reconstruct and display the modified paragraph

END