PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Chitra G M
Computer Science and Engineering

PES UNIVERSITY

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Introduction

Chitra G M

Department of Computer Science and Engineering

OVERVIEW OF THE COURSE

- Learn Computational modes of thinking.
- Master the art of computational problem solving.
- Make computers do what you want to do.

Topics

- Control Structures.
- Data Structures, Files.
- Functions.
- Functional Programming.
- OOP.

What does a computer do?

- Fundamentally:
  - Performs Calculations.
  - Remembers results.

What type of calculations a computer can do ?

- Set of Built in oprations. Typically arithmetic, and simple logic operations.

- Create a new operation.

Can a computer perform / solve any task that exists?

The task or job can be either :

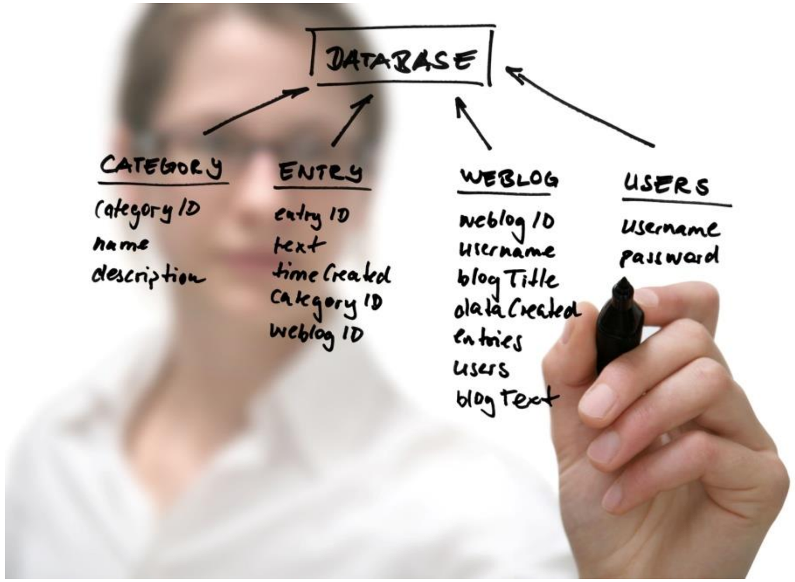
- Computational: The problems that can be solved

- Non-Computational: The problem that can not be solved.

Two things that are needed to perform Computational problem solving:

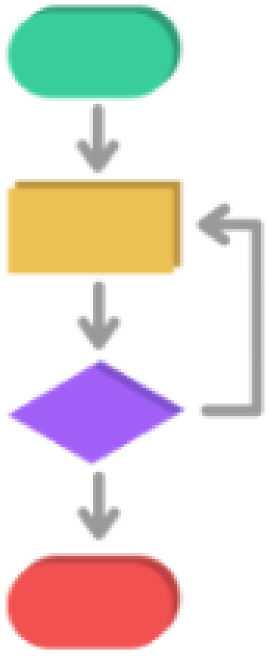
- **A representation** that captures all the relevant aspects of the problem

- **An algorithm** that solves the problem by use of the representation.



Algorithm:

An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.



Algorithm

The word “algorithm” is derived from the ninth-century Arab mathematician, Al-Khwarizmi.

Man, Cabbage, Goat, Wolf Problem.

A man lives on the east side of a river. He wishes to bring a cabbage, a goat, and a wolf to a village on the west side of the river to sell.

Example: Man, Cabbage, Goat, Wolf Problem.

However, his boat is only big enough to hold himself, and either the cabbage, goat, or wolf. In addition, the man cannot leave the goat alone with the cabbage because the goat will eat the cabbage, and he cannot leave the wolf alone with the goat because the wolf will eat the goat.

How does the man solve his problem?

Example: Video to visualize the solution

Example: Man, Cabbage, Goat, Wolf Problem.

Solution:

There is a simple algorithmic approach for solving this problem by simply trying all possible combinations of items that may be rowed back and forth across the river.

Trying all possible solutions is referred to as a **brute force approach**.

Example: Man, Cabbage, Goat, Wolf Problem.

The computational problem is to find a way toconvert the representation of the start state of the problem, when all the object are on the east side of the river,

man cabbage goat   wolf

[E, E, E, E]

to the goal state with all objects on the west side of the river.

Example: Man, Cabbage, Goat, Wolf Problem.

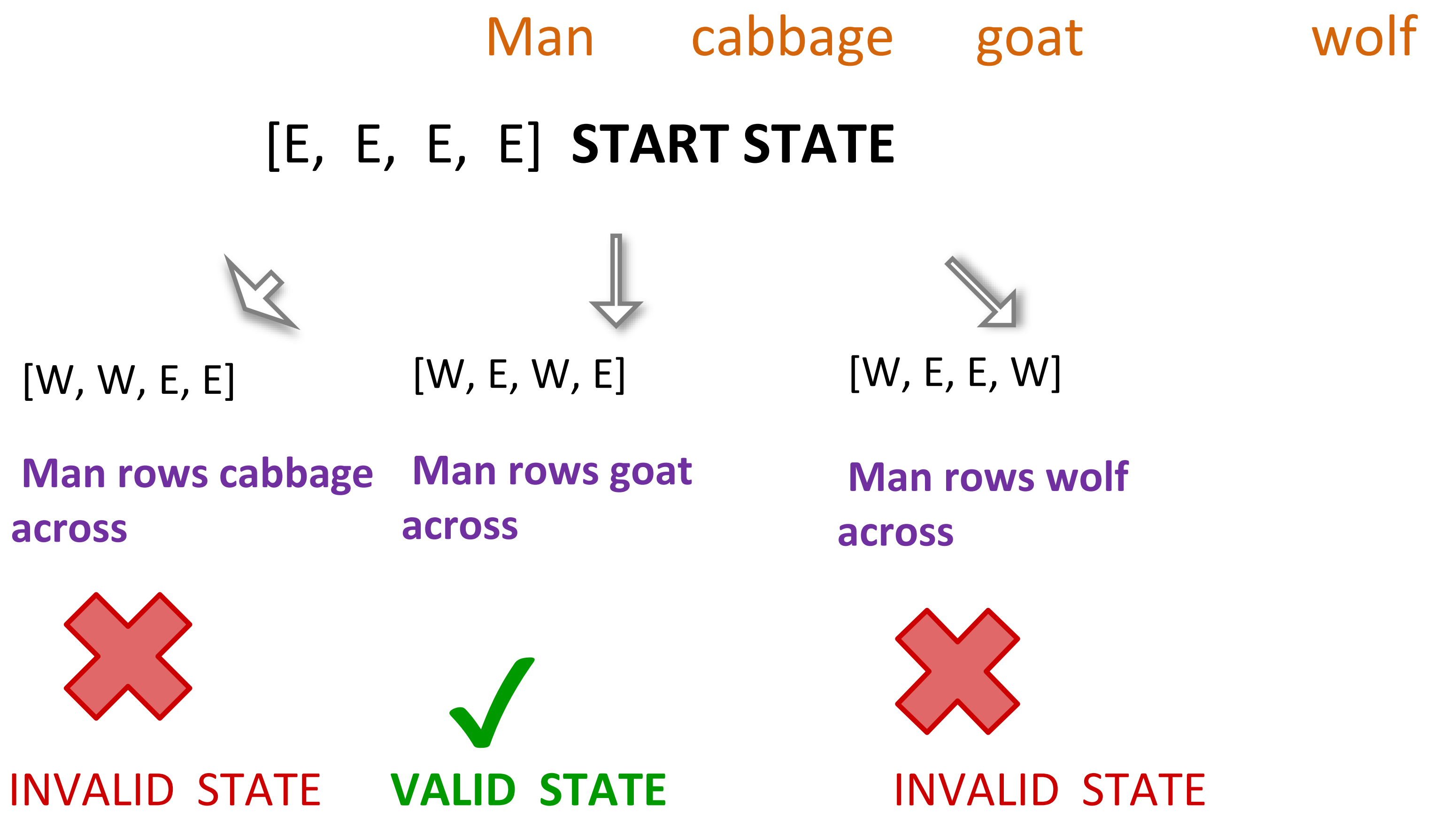
Man cabbage goat wolf

[W, W, W, W]

with the constraint that certain invalid states should never be used.

Example: Man, Cabbage, Goat, Wolf Problem.

For example, from the start state, there are three possible moves that can be made, only one of which results in a valid state.



Man cabbage goat wolf

[E, E, E, E] **START STATE**

[W, W, E, E]

Man rows cabbage across

✗

INVALID STATE

[W, E, W, E]

Man rows goat across

✓

VALID STATE

[W, E, E, W]

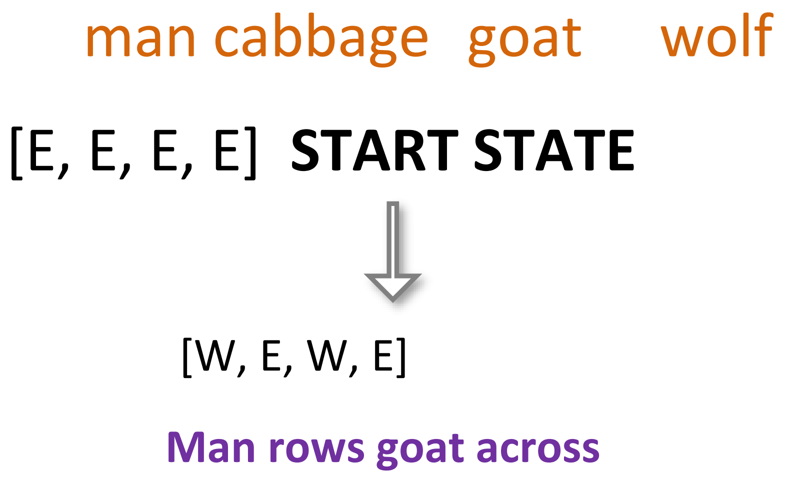
Man rows wolf across

✗

INVALID STATE

Example: Man, Cabbage, Goat, Wolf Problem.

We check if the new problem state is the goal state. If true, then we solved the problem in one step! (We know that cannot be so, but the algorithmic approach that we are using does not.)



man cabbage goat wolf

[E, E, E, E] **START STATE**

[W, E, W, E]

Man rows goat across

Is goal state [W,W,W,W]? **NO**

Therefore we continue searching from the current state.

Example: Man, Cabbage, Goat, Wolf Problem.

Since the man can only row across objects on the same side of the river, there are only two possible moves from here,

man   cabbage   goat   wolf   **INTERMEDIATE**

[W, E, W, E]   **STATE**

[E, W, E, E]

Man rows back alone

✓

[E, E, E, E]

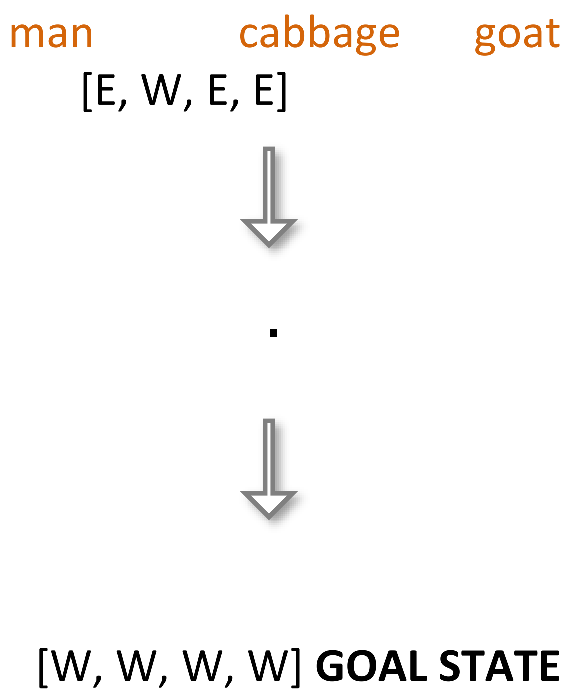
Man rows goat across

✓

VALID STATES

Example: Man, Cabbage, Goat, Wolf Problem.

This would continue until the goal state is reached,



man cabbage goat wolf

[E, W, E, E]

↓

.

↓

[W, W, W, W] **GOAL STATE**

Example: Man, Cabbage, Goat, Wolf Problem.

Thus, **the computational problem of generating the goal state from the start state translates into a solution of the actual problem since each transition between states has a corresponding action in the actual problem**—of the man rowing across the river with (or without) a particular object.

THANK YOU

Chitra G M

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Chitra G M and Gayathri R S

Computer Science and Engineering

Topics covered

- Digital Computer
- Computer Hardware
- Operating System
- Computer Software
- Syntax, semantics and program translation

Digital Computer

The Digital computer is the most commonly used type of computer and is used to process information with quantities using digits, usually using the binary number system.

Number System

- A number system is defined as the representation of numbers by using digits or other symbols in a consistent manner.

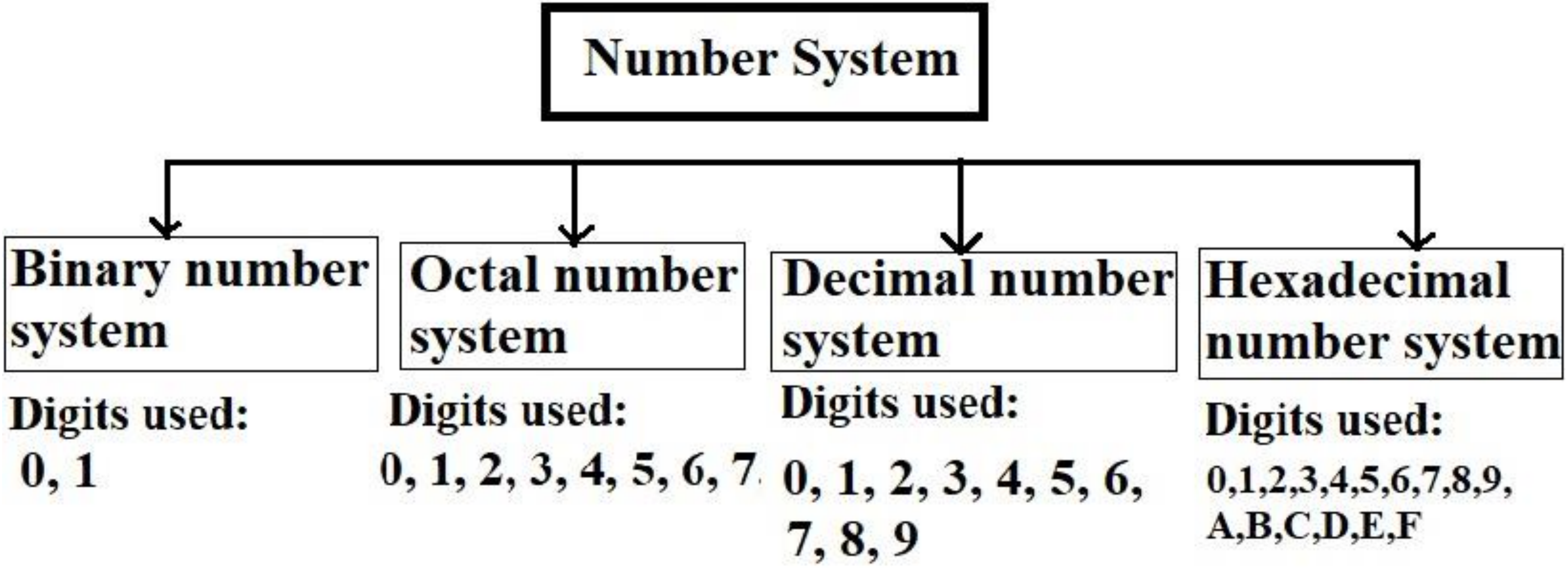
- The value of any digit in number can be determined by a digit, its position in the number, and the base of the number system.

Number System

There are different types of number systems in which the four main types are:

- **Binary number system (Base - 2)**
- **Octal number system (Base - 8)**
- **Decimal number system (Base - 10)**
- **Hexadecimal number system (Base - 16)**

Number System



Number System

Binary number system

Digits used:
0, 1

Octal number system

Digits used:
0, 1, 2, 3, 4, 5, 6, 7

Decimal number system

Digits used:
0, 1, 2, 3, 4, 5, 6,
7, 8, 9

Hexadecimal number system

Digits used:
0,1,2,3,4,5,6,7,8,9,
A,B,C,D,E,F

Number System – Binary Number System

- **Binary number system (Base - 2)**

  For representing numbers in **base 2**, there are two possible digits (0, 1) in which each column value is a power of two*:*

128	64	32	16	8	4	2	1
0	1	1	0	0	0	1	1

0 + 64 + 32 + 0 + 0 + 0 + 2 + 1 = 99

Computer Hardware

Computer hardware comprises the physical part of a computer system. It includes :

- **central processing unit** (CPU) and **main memory**
- **peripheral components** such as a keyboard, monitor, mouse, and printer.

Computer Hardware

Central processing unit (CPU) – the “brain” of a computer system. Interprets and executes instructions.

Main memory – is where currently executing programs reside.It is volatile, the contents are lost when the power is turned off.

Secondary memory – provides long-term storage of programs and data. Non-volatile, the contents are retained when power is turned off. Can be magnetic (hard drive), optical (CD or DVD), or flash memory (USB drive).

Computer Hardware

Input– mouse, keyboard, scanner, microphone etc.

Output Devices – monitor, printer, projector, speakers etc

Computer Hardware

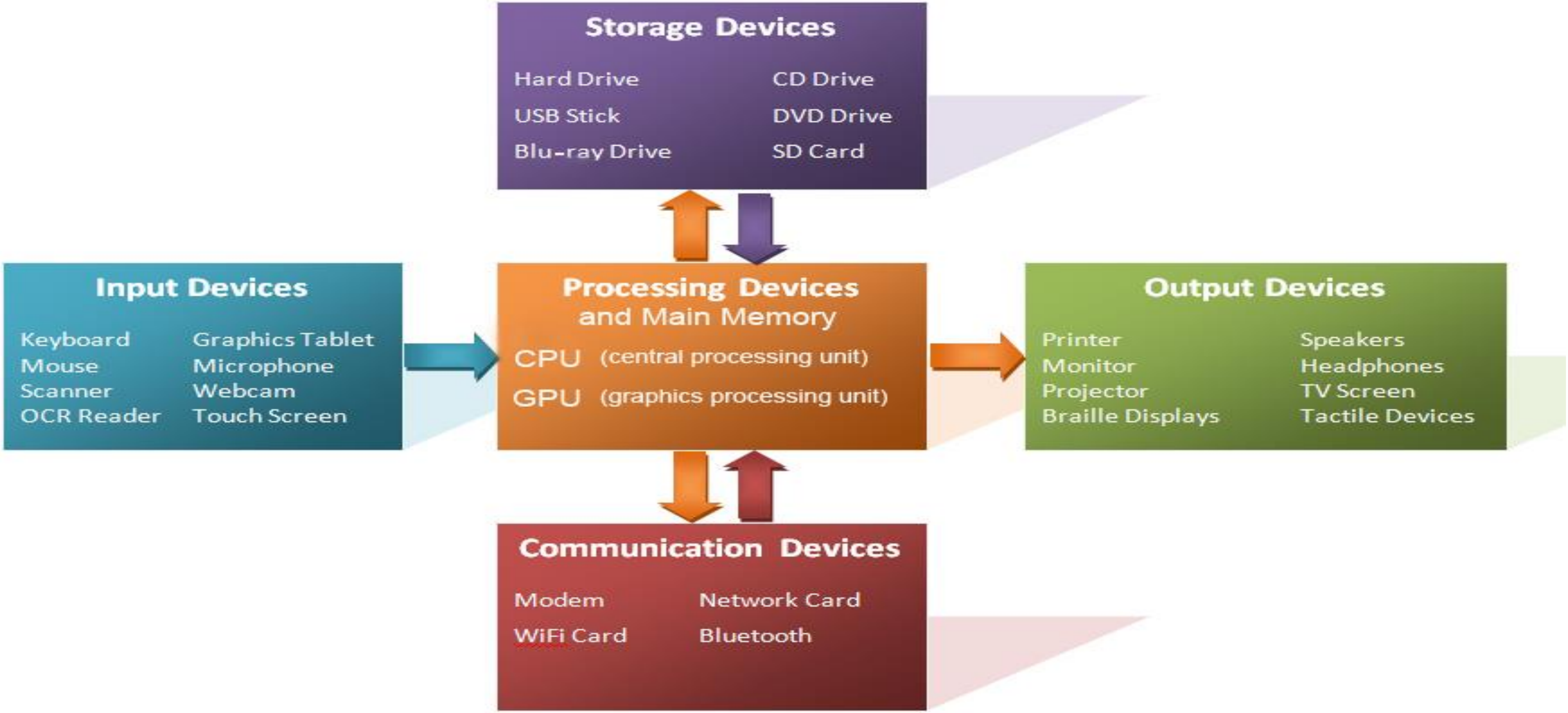
Buses – is a communication system that transfers data between components inside a computer , or between computers.

- Internal Bus (System Bus: CPU and Main Memory)

-External Bus (Expansion Bus :printer to the computer)

Communication devices- modem, WiFi card, etc

Computer Hardware



Storage Devices

Hard Drive
USB Stick
Blu-ray Drive
CD Drive
DVD Drive
SD Card

Input Devices

Keyboard
Mouse
Scanner
OCR Reader
Graphics Tablet
Microphone
Webcam
Touch Screen

Processing Devices and Main Memory

CPU (central processing unit)
GPU (graphics processing unit)

Output Devices

Printer
Monitor
Projector
Braille Displays
Speakers
Headphones
TV Screen
Tactile Devices

Communication Devices

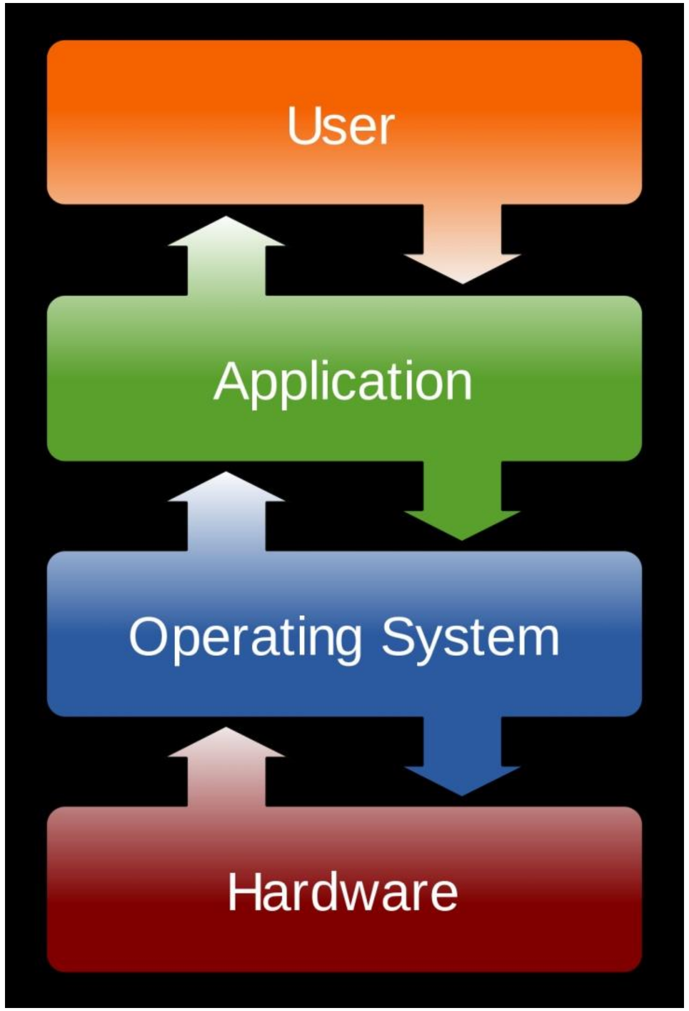
Modem
WiFi Card
Network Card
Bluetooth

Operating System

An **operating system** is software that manages and interacts with the hardware resources of a computer.

Because an operating system is intrinsic to the operation of a computer, it is referred to as **system software**.

Operating System



Computer Software

Computer software is a set of program instructions, including related data and documentation, that can be executed by computer.

- **System software:** intrinsic to a computer system.
- **Application Software:** Application software is specific purpose software which is used by user for performing specific task.

Computer Software

- The first computer programs ever written were for a mechanical computer designed by **Charles Babbage** in the mid-1800s.

- **Ada Lovelace** was the person who wrote these programs.

- she is referred to as **"the first computer programmer."**

Ada Lovelace

System Software vs Application Software

S.No.	System Software	Application Software
1.	System software is used for operating computer hardware.	Application software is used by user to perform specific task.
2.	System software are installed on the computer when operating system is installed.	Application software are installed according to user's requirements.
3.	In general, the user does not interact with system software because it works in the background.	In general, the user interacts with application software.
4.	System software can run independently. It provides platform for running application softwares.	Application software can't run independently. They can't run without the presence of system software.
5.	Some examples of system softwares are compiler, assembler, debugger, driver, etc.	Some examples of application software's are word processor, web browser, media player, etc.

Syntax

English, for example, includes the letters of the alphabet, punctuation, and properly spelled words and properly punctuated sentences.

The **syntax** of a language is a set of characters and the acceptable sequences (arrangements) of those characters.

The following is a syntactically correct sentence in English,

I will read tomorrow

The following, however, is not syntactically correct,

I tomorrow reed will

Semantics

Consider the following sentence:

I will read yesterday

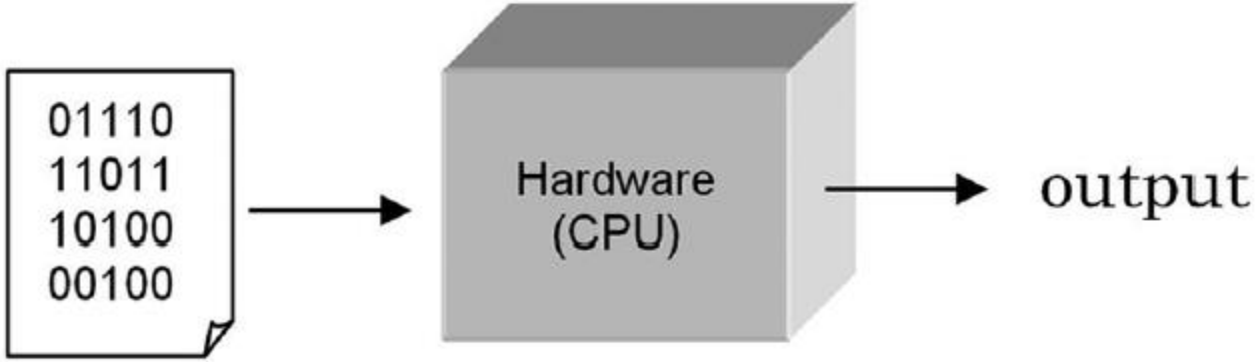
This sentence is syntactically correct, but has no meaning. Thus, it is *semantically incorrect.*

The **semantics** of a language is the meaning associated with each syntactically correct sequence of characters.

Every language has its own syntax and semantics.

Program Translation.

A central processing unit (CPU) is designed to interpret and execute a specific set of instructions represented in binary form (i.e., 1s and 0s) called **machine code**. Only programs in machine code can be executed by a CPU.



Program Translation.

Writing programs at this “low level” is tedious and error-prone.

Therefore, most programs are written in a “high-level” programming language such as Python.

Since the instructions of such programs are not in machine code that a CPU can execute, a **translator** program must be used.

Program Translation.

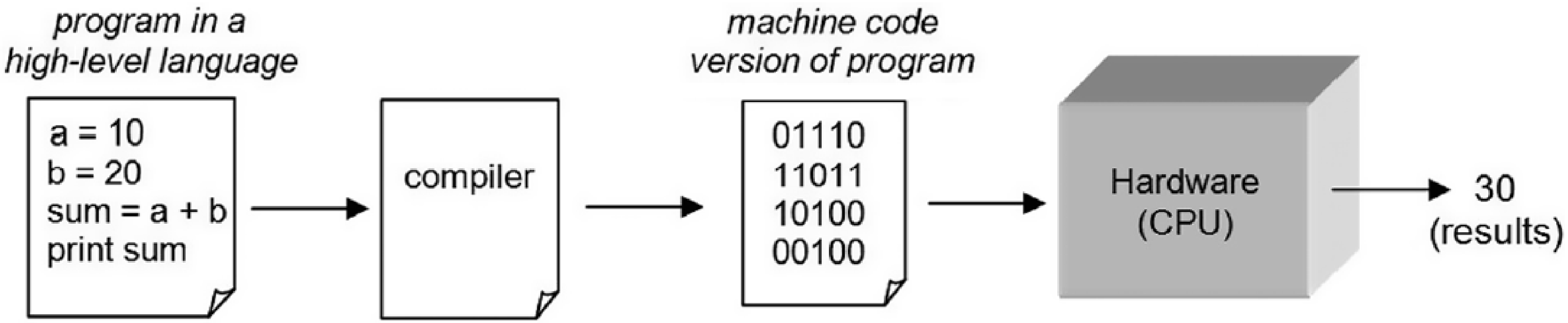
There are two fundamental types of translators:

1. **Compiler**
2. **Interpreter**

Both compilers and interpreters are software that convert a code written in a high-level language into a lower-level or machine code understood by computers.

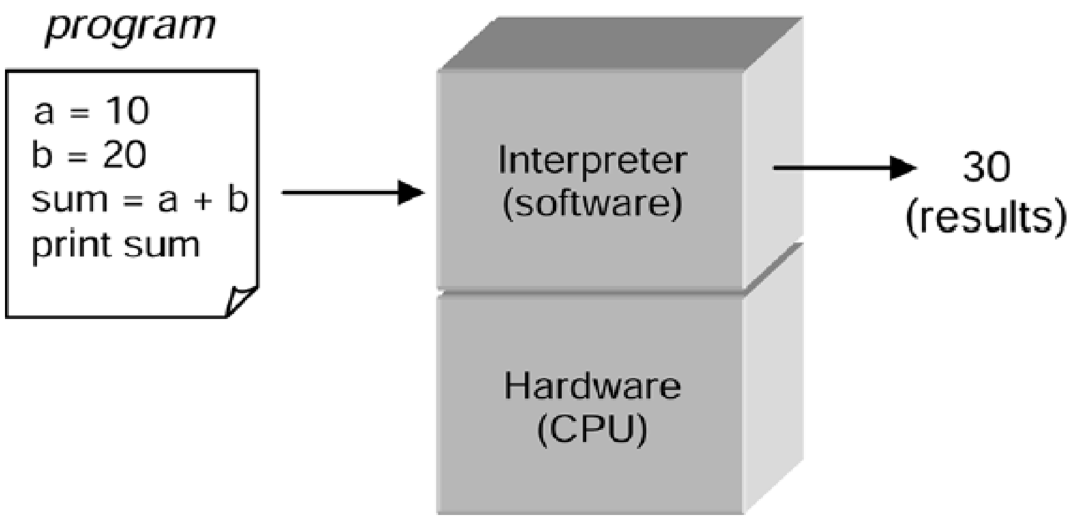
Compiler

A compiler translates code from a high-level programming language into machine code before the program runs.



Interpreter

An interpreter translates code written in a high-level programming language into machine code line-by-line as the code runs.



program

```
a = 10
b = 20
sum = a + b
print sum
```

Difference between Compiler and Interpreter

Compiler	Interpreter
Scans the entire program and converts it to machine code as a whole.	Translates one statement at a time.
Compiler produces an executable file that can be executed multiple times without requiring recompilation.	An interpreter does not produce any standalone executable file
Target programs run on their own. They don’t need the Compiler in memory to work.	At the time of interpretation, the interpreter resides in the memory.
Used in programming languages like C, C++, and Java	Used in languages like JavaScript, Python, and Ruby.

THANK YOU

Chitra G M, Gayathri R S

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Prof. Sindhu R Pai

Computer Science and Engineering

Intro to Translation models

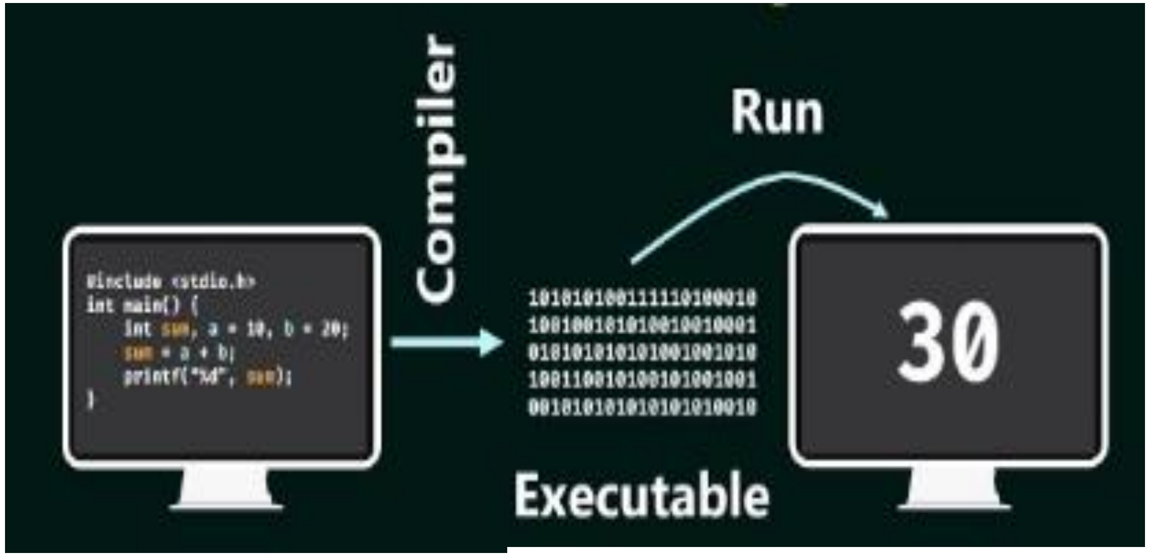
- A programming language provides the necessary constructs to instruct the computer to do the tasks for us
- Levels – High, medium, low
- Generations – 1 to 5
- Translation models describe the mathematical relationship between two or more languages.
    - **Models of translational equivalence** - Whether expressions in different languages have equivalent meanings.

Language Translators

- **Compilers, Interpreters, and Assemblers, …**
- Compiler converts entire high-level language programs to machine language at once.
- Interpreter converts high-level language to machine language line by line.

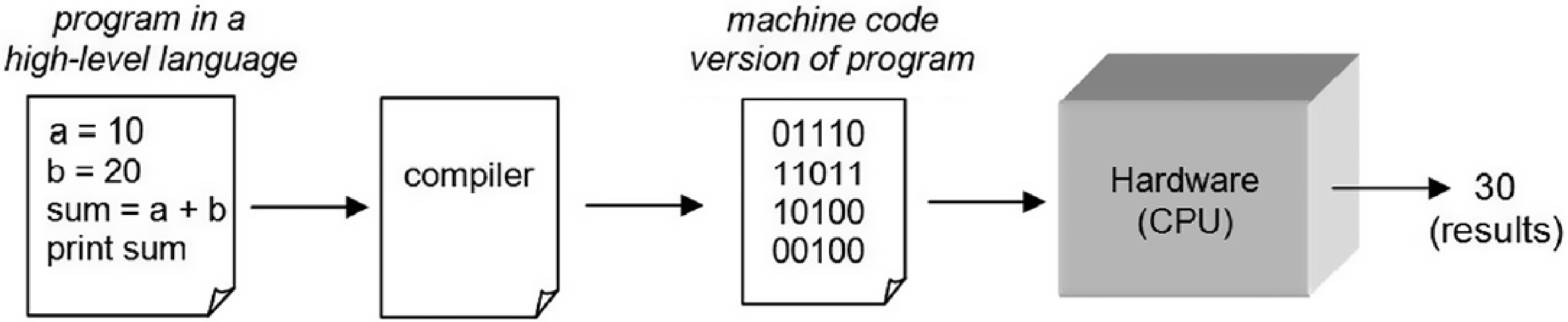
PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Compiler



```
#include <stdio.h>
int main() {
    int sum, a = 10, b = 20;
    sum = a + b;
    printf("%d", sum);
}
```

```
1010101001111110100010
1001001010100100010001
01010101010101001001010
100110010100101001001
001010101010101010010
```



program in a high-level language

```
a = 10
b = 20
sum = a + b
print sum
```

compiler

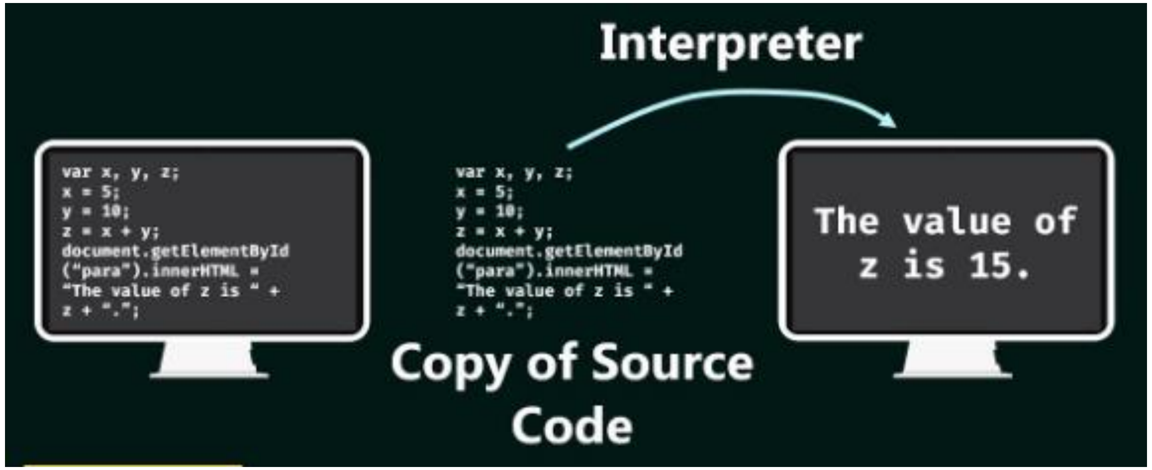
machine code version of program

```
01110
11011
10100
00100
```

Hardware (CPU)

30 (results)

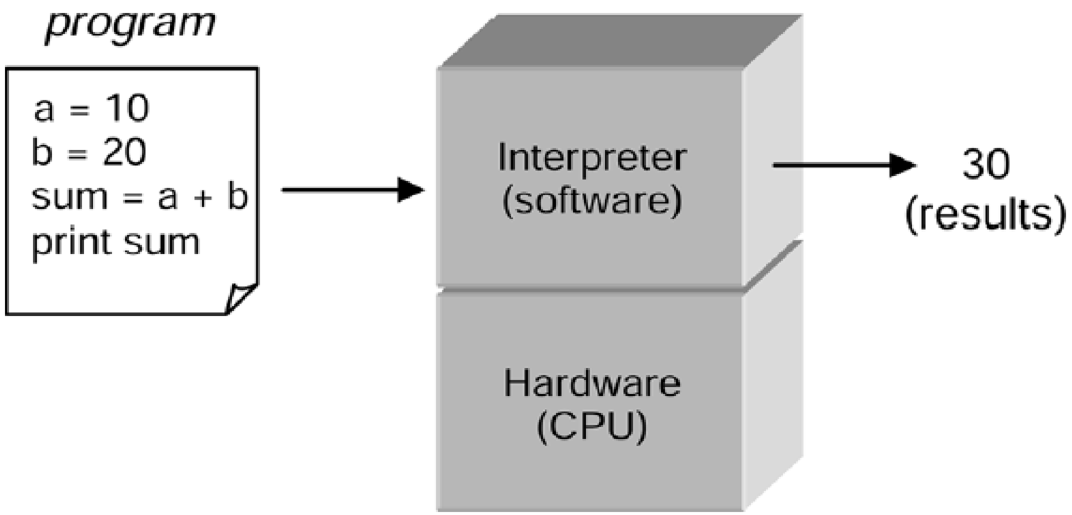
Interpreter



```
var x, y, z;
x = 5;
y = 10;
z = x + y;
document.getElementById
("para").innerHTML =
"The value of z is " +
z + ".";
```

```
var x, y, z;
x = 5;
y = 10;
z = x + y;
document.getElementById
("para").innerHTML =
"The value of z is " +
z + ".";
```

```
The value of
z is 15.
```



```
a = 10
b = 20
sum = a + b
print sum
```

PES UNIVERSITY

THANK YOU

sindhurpai@pes.edu

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Chitra G M

Computer Science and Engineering

Topics covered

Process of Computation Problem Solving

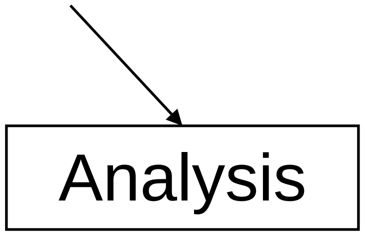
- Analysis,
- Design,
- Implementation &
- Testing

Process of Computation Problem Solving

- **Computational problem solving** does not simply involve the act of computer programming.

- It is a *process*, with programming being only one of the steps.
- Before a program is written, a design for the program must be developed.
- And before a design can be developed, the problem to be solved must be well understood.
- Once written, the program must be thoroughly test

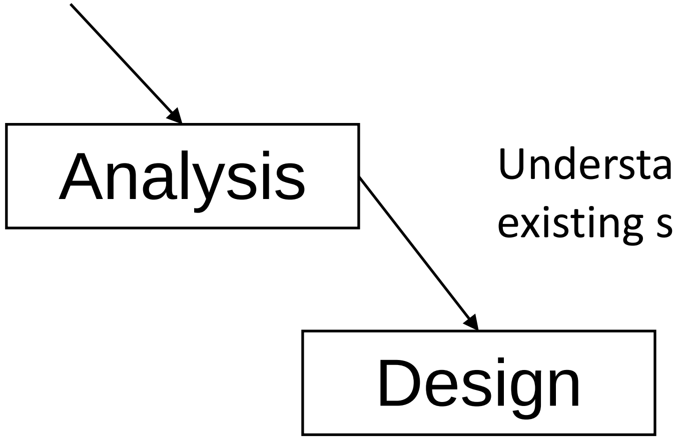
Process of Computation Problem Solving



Analysis

Understand the problem clearly and understand the existing solutions if any for the given problem

Process of Computation Problem Solving



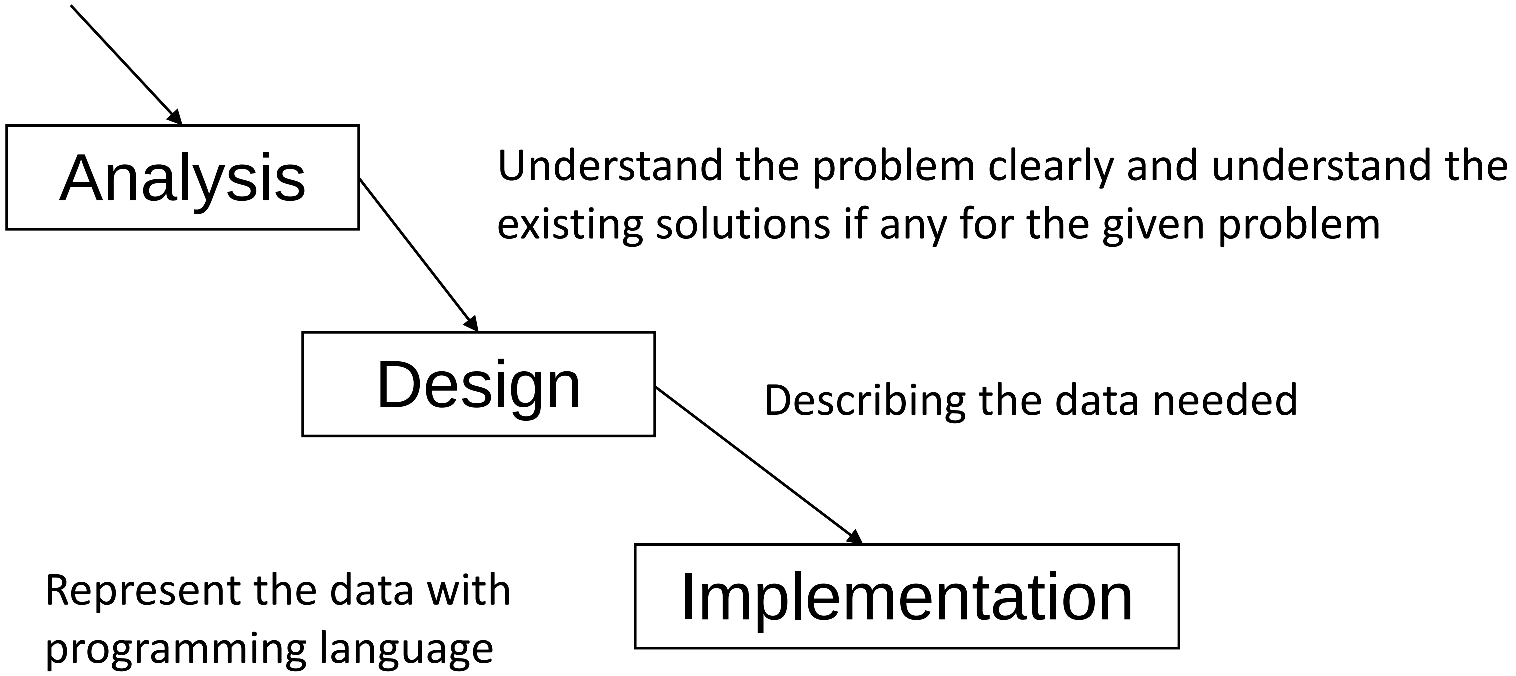
Analysis

Understand the problem clearly and understand the existing solutions if any for the given problem

Design

Describing the data needed

Process of Computation Problem Solving

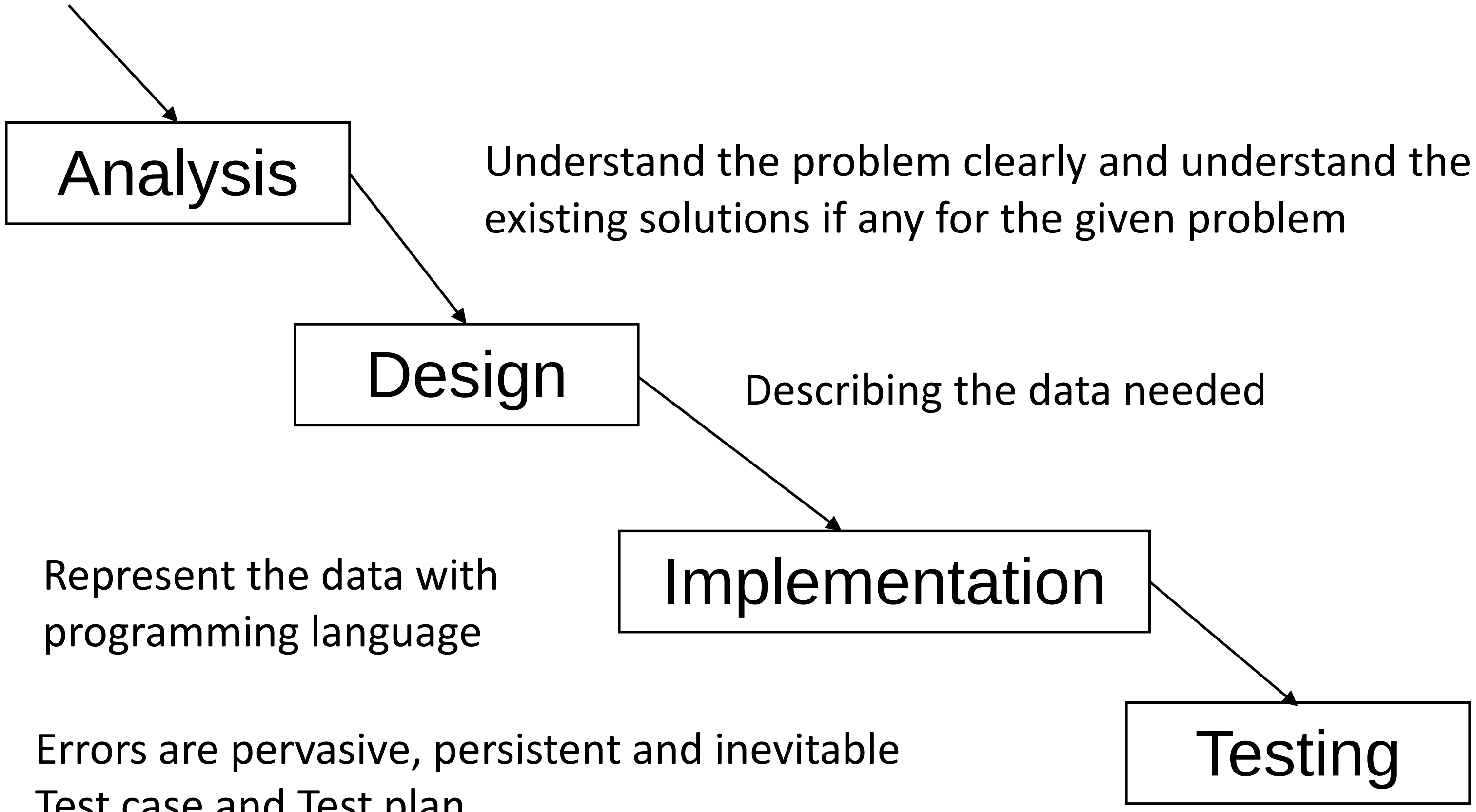


Understand the problem clearly and understand the existing solutions if any for the given problem

Describing the data needed

Represent the data with programming language

Process of Computation Problem Solving



Analysis

Understand the problem clearly and understand the existing solutions if any for the given problem

Design

Describing the data needed

Implementation

Represent the data with programming language

Testing

Errors are pervasive, persistent and inevitable
Test case and Test plan

Process of Computation Problem Solving

Problem Analysis:

- Must understand the fundamental computational issues involved.

- Example:

  For MCGW problem, can use brute-force approach of trying all of the possible rowing actions that may be taken

Knowing what constitutes a solution.

For some problems, there is only one solution. For others, there may be a number (or infinite number) of solutions. Thus, a problem may be stated as finding,

- **A solution**
- **An approximate solution**
- **A best solution**
- **All solutions**

Process of Computation Problem Solving

Describe Data and Algorithms

- For the **MCGW problem**, need to store the current state of the problem.

- When solving a computational problem, either suitable existing algorithms may be found, or new algorithms must be developed.

Program Implementation

- Design decisions provide general details of the data representation and the algorithmic approaches for solving a problem.

- The details, however, do not specify which programming language to use, or how to implement the program.

- That is a decision for the implementation phase.

Program Implementation

Since we are programming in Python, the implementation needs to be expressed in a syntactically correct and appropriate way, using the instructions and features available in Python.

Program Testing

Writing computer programs is difficult and challenging. As a result, **programming errors are pervasive, persistent and inevitable**.

Given this fact, **software testing is a crucial part of software development**. Testing is done incrementally as a program is being developed, when the program is complete, and when the program needs to be updated.

Facts of Software Development

- Programming errors are pervasive, persistent, and inevitable.
- Software testing is an essential part of software development.
- Any changes made in correcting a programming error should be fully understood as to why the changes correct the detected error.

THANK YOU

Chitra G M

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Chitra G M

Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Introduction to the Python Programming environment

Modes of Programming

- Interactive mode
  - **python**
  - >>> <enter command here>

- Batch mode
  - Create a file with python commands
  - Run them together
  - **python filename.py**

Program Structure

- Sequential execution
- Comments
- Execution starts from the 1st column
- Case Sensitive
- Syntax
- Number of Statements per line
- No free flow code

Program Structure

- **Comments:**
  - Single line comments begin with a # symbol
  - The documentation string is used for commenting multiple lines which is written using a pair of triple " " " " " " quotes or ' ' ' ' ' '
- **Case sensitive**
  - **Print()** and **print()** are different

Program Structure

- **Number of Statements per line**
  - Ideally there must be only One statement per line
  - One statement can pan across multiple lines
    1. Use Escape character ‘\’ to ignore the EOL
    2. Use ( ) constructs
  - Multiple statements in one line separated by a ‘ ; ’

THANK YOU

Chitra G M

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Chitra G M

Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Output Function

Output Function

The **print** function

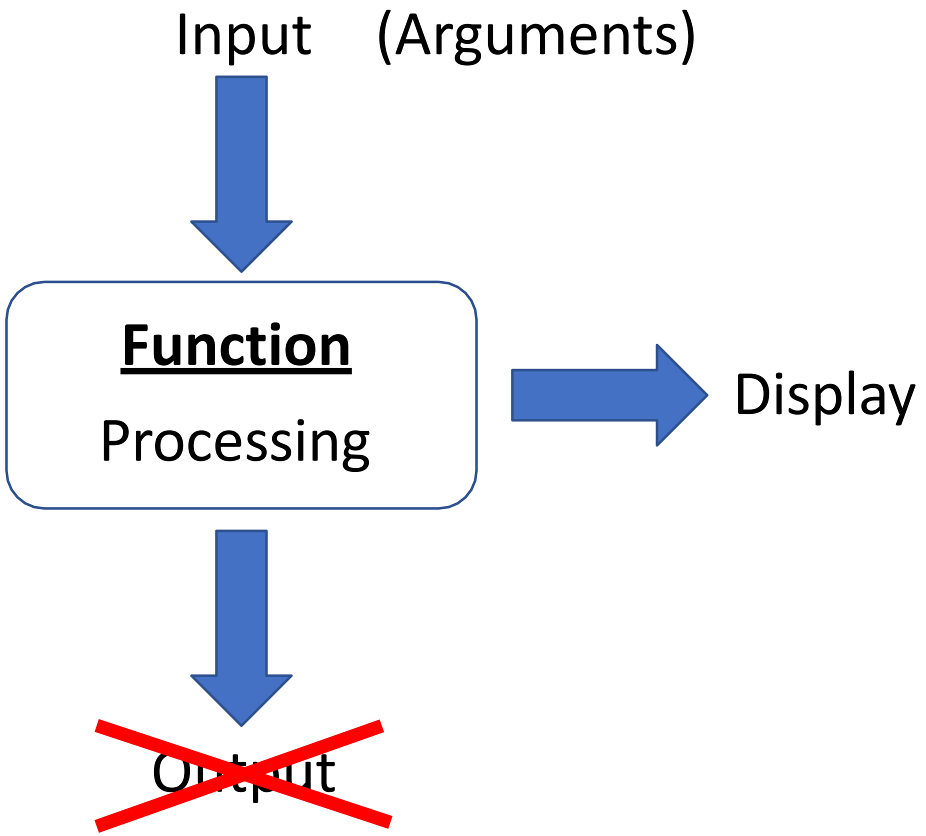
1. Types of arguments

2. Number of arguments

3. Configuring the print function

Output Function

The print function



Types of Arguments

- Can take any value as arguments
- Values : Numbers, Boolean, strings, collections, expressions, functions, etc.
- Expressions are evaluated and the result is displayed

Number of Arguments

- Can take any number of arguments

- Displayed with a default field separator : (space) ‘ ’

print Function Definition

print(...)

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

- **file**: a file-like object (stream); defaults to the current sys.stdout.
- **sep**: string inserted between values, default a space.
- **end**: string appended after the last value, default a newline.
- **flush**: whether to forcibly flush the stream

Configuring the Space between the output fields

- By default, the output field separator is a space character

```
>>> print(10,12,14)
10 12 14
```

- To change the output field separator, use the **sep** argument

```
>>> print(10,12,14, sep = “:”)
10:12:14
```

Configuring the output record separator

- By default, the output record separator is a new line character

Example:

```
>>> print(10,12,14);print(16,18)
10 12 14
16 18
```

- To change the output record separator,

  use the **end** argument

```
>>> print(10,12,14, end = "");print(16,18)
10 12 14 16 18
```

THANK YOU

Chitra G M

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Chitra G M, Gayathri R S

Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Variables and Identifiers

What Is an Identifier?

- An **identifier** is a sequence of one or more characters used to provide a name for a given program element.

  Examples: name , srn_number, ph_no, marks1, marks2

- It is used to identify the program element

- Python is **case sensitive** , thus, `name` is different from `Name`

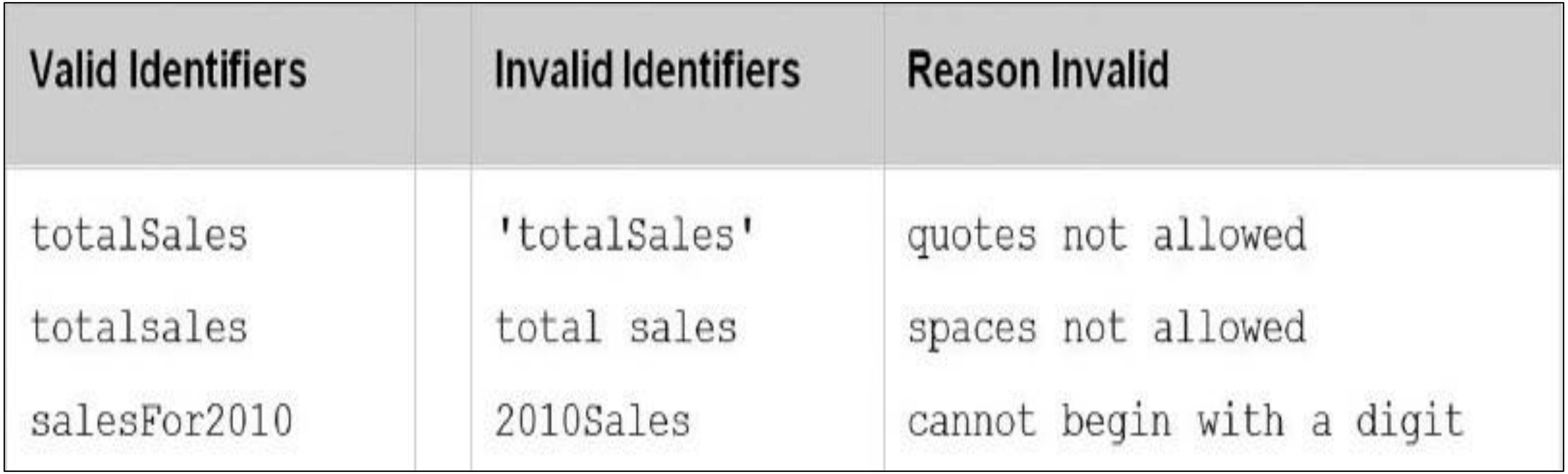
- Identifiers may contain letters and digits and underscore characters

Identifiers

Naming Convention:

1. Can begin with alphabets a-z or A-Z
2. Cannot begin with a digit 0-9 or a special character
3. Spaces are not allowed as part of an identifier
4. underscore character, _ is also allowed to aid in the readability of long identifier names. The variables that begin with underscore has a special meaning in object oriented programming. So we do not prefer to use _ as the first character.
5. Keywords cannot be used as variables.
6. Quotes are not allowed.

Valid and invalid Identifiers



Valid Identifiers		Invalid Identifiers	Reason Invalid
totalSales		'totalSales'	quotes not allowed
totalsales		total sales	spaces not allowed
salesFor2010		2010Sales	cannot begin with a digit

Keywords

- Keywords are **reserved** words that have a predefined meaning.
- To know the keywords, execute the below statement
- print(help('keywords'))

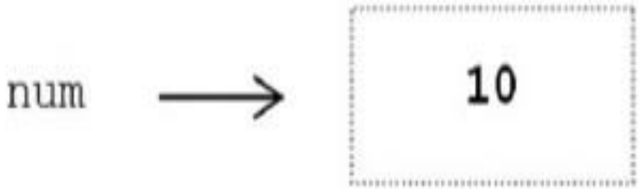
Keywords

List of keywords in python:

False
None
True
and
as
assert
break
class
continue
def
del
elif
else
except
finally
for
from
global
if
import
in
is
lambda
nonlocal
not
or
pass
raise
return
try
while
with
yield

What Is a Variable?

A **variable** is a name (identifier) that is associated with a value.



variable `num` is assigned the value 10

A variable can be assigned different values during a program’s execution—hence, the name “variable.”

Variables

One of the most fundamental concepts in programming is that of a **variable**.

Whenever variable **num** appears in a calculation, it is the current value of **num** that is used

```
num + 20    (10  + 20)
```

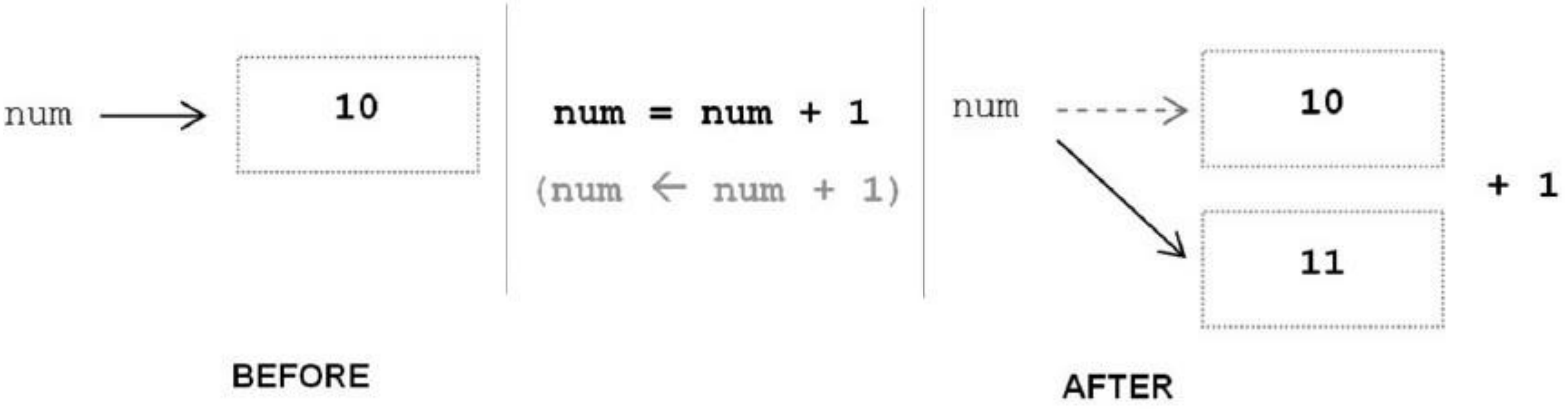
If variable **num** is assigned a new value, then the same expression will produce a different result

```
num = 5
num + 20    (5 + 20)
```

Variables

Variables are assigned values by use of the **assignment operator** **, =**

num = 10

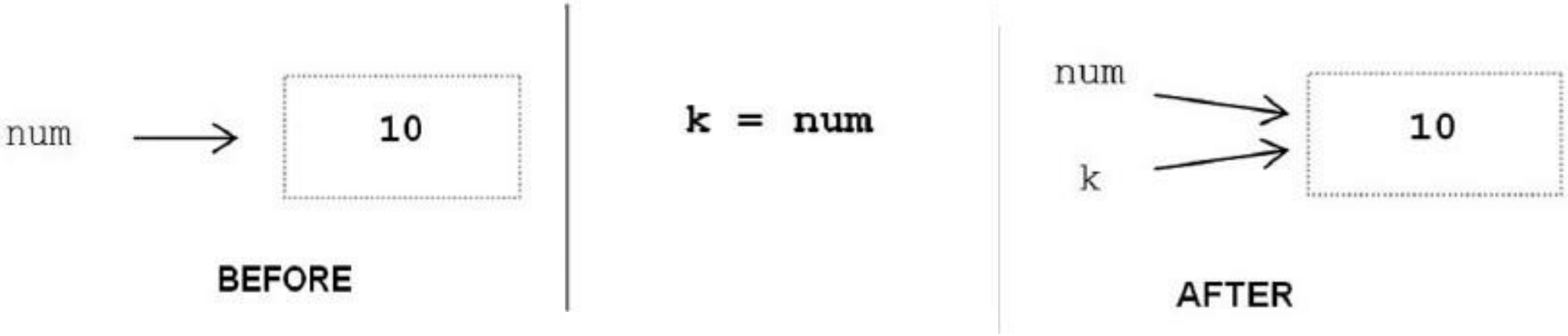


the right side of an assignment is evaluated first, then the result is assigned to the variable on the left.

Variables

Variables may also be assigned to the value of another variable.

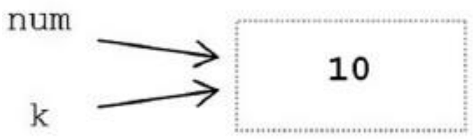
num = 10



Variables **num** and **k** are both associated with the same literal value 10 in memory. One way to see this is by use of **built-in function** id

Variables

If the value of **num** changed, would variable **k** change along with it?

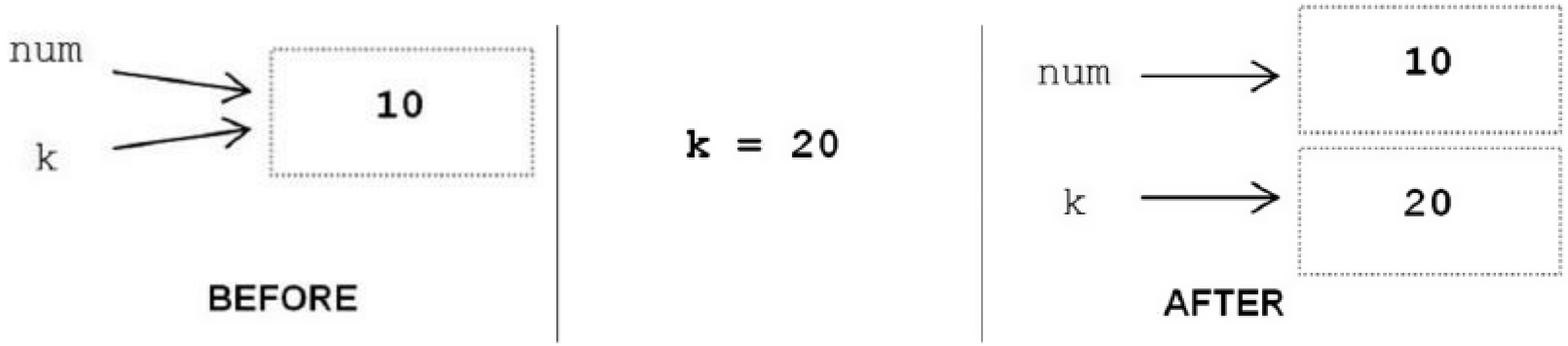


Here variables refer to integer values, and integer values are *immutable.*

An **immutable value** is a value that cannot be changed.

Thus, both will continue to refer to the same value until one (or both) of them is reassigned

Variables



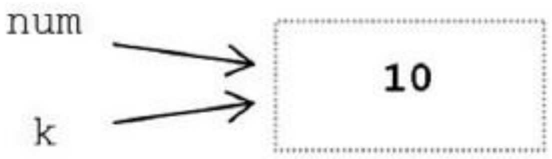
If no other variable references the memory location of the original value, the memory location is ***deallocated*** *(that is, it is made available for reuse).*

id function

- the id() function is a built-in function that returns the identity of an object.
- The id() function is commonly used to check if two variables or objects refer to the same memory location.
- The **is** keyword is used to test whether two variables belong to the same object. The test will return **True** if the two objects are the same else it will return **False**

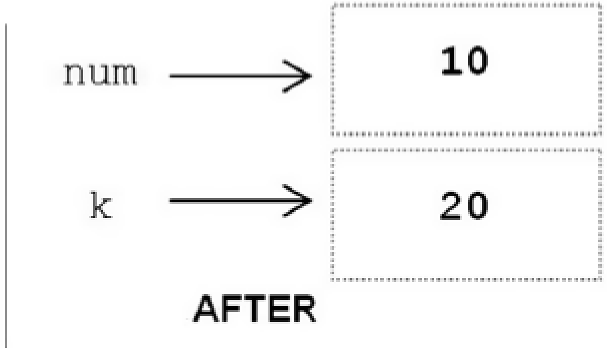
id function

```
>>> num=10
>>> k=10
>>> id(num)
2863970058768
>>> id(k)
2863970058768
>>> num is k
True
>>> k=20
>>> id(num)
2863970058768
>>> id(k)
2863970059088
>>> num is k
False
```



BEFORE

k = 20



AFTER

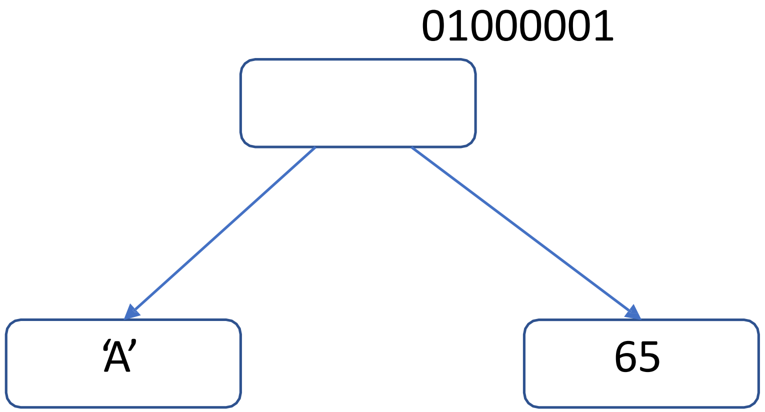
Data Types

- Datatype refers to the type of value a variable has.

- Integers, floats, and strings are part of a set of predefined data types in Python called the **built-in types**

Data Types

The need for data types results from the fact that the same internal representation of data can be interpreted in various ways



The sequence of bits in the figure can be interpreted as a character ('A') or an integer (65).

Data Types

- There are two approaches to data typing in programming languages:
  - **Static**
  - **Dynamic**

- **Static typing**, a variable is declared as a certain type before it is used, and can only be assigned values of that type.

- **Dynamic typing**, the data type of a variable depends only on the type of value that the variable is currently holding
  - the same variable may be assigned values of different type during the execution of a program

The function type()

A built-in function, that returns the type of the object

Syntax:

type(object)

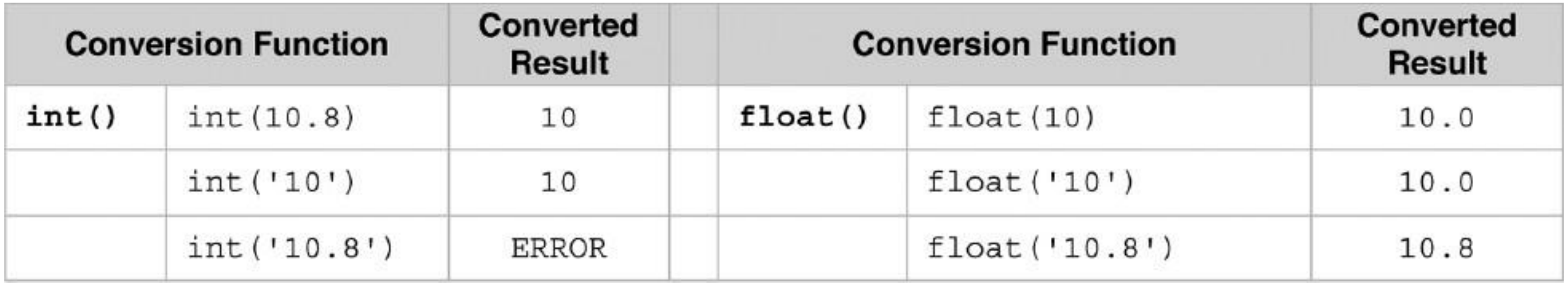
Type of a variable depends on the value assigned to it a = 10

```
print(type(a))  #int
a = 10.0
print(type(a)) # float
```

Type conversion functions int() and float()

Note that numeric strings can also be converted to a numeric type.

num_credits = int(input('How many credits do you have? '))



Conversion Function		Converted Result		Conversion Function		Converted Result
int()	int(10.8)	10		**float()**	float(10)	10.0
	int('10')	10			float('10')	10.0
	int('10.8')	ERROR			float('10.8')	10.8

THANK YOU

Chitra G M , Gayathri R S

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

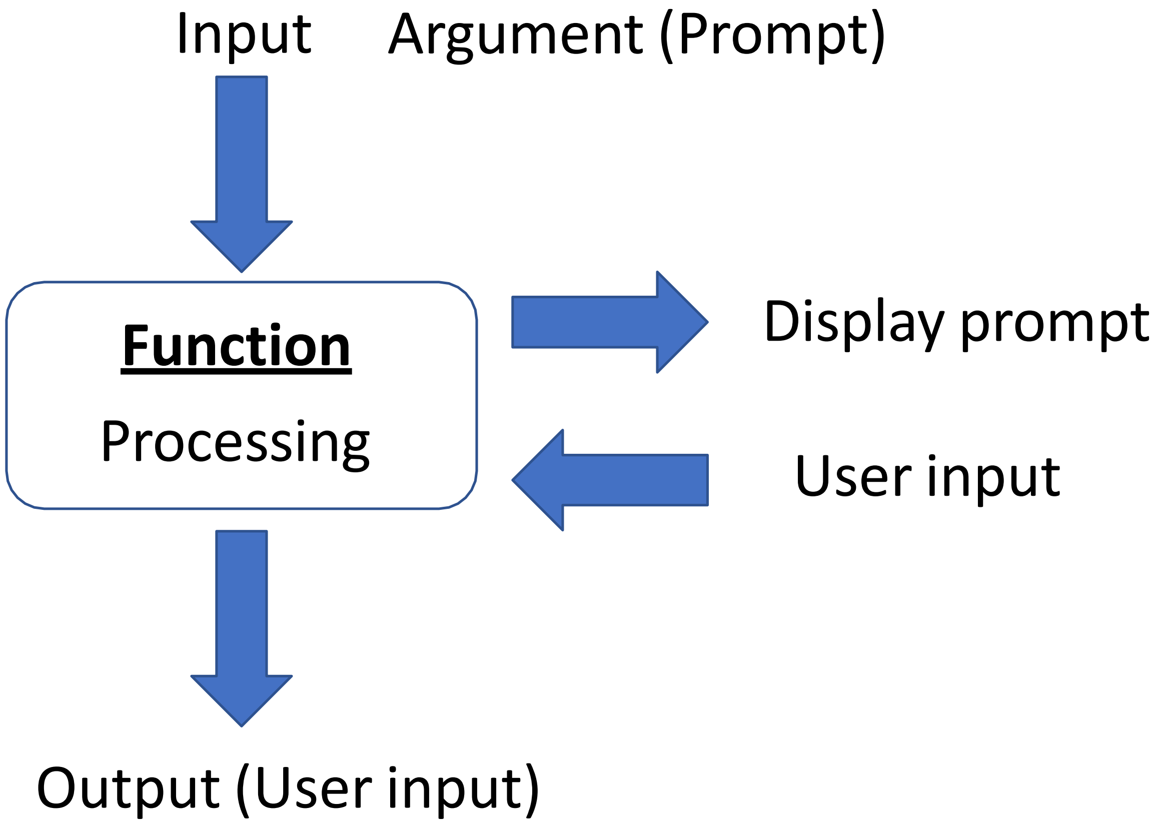
Prof. Gayathri R S

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Input Function

The input function


Input Argument (Prompt)

Function

Processing

Display prompt

User input

Output (User input)

Input Function Definition

In python , 3.x and above

```
input(prompt=None, /)
    Read a string from standard input.  The trailing newline is stripped.

    The prompt string, if given, is printed to standard output without a trailing
    newline before reading input.


    If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOF Error.
```

Input Function

Syntax

input([prompt])

- The prompt is a string that would be displayed in the terminal
- It is an optional argument
- Since the input function returns a value, it must be stored in a variable for future use

Example:

length = input(“Enter the length in cm: ”)

Return Value

- The value **returned** by the input function is a **string**

- **Type conversion** must be used if required.

```
>>> length = input("Enter the length in cm: ")
Enter the length in cm: 10
>>> type(length)
<class 'str'>
>>> length=int(length)
>>> type(length)
<class 'int'>
```

Multiple values as input

- We can use the split function that is available for strings
- split() returns a list of strings

```
>>> input("Enter maths and science marks:  ")
Enter maths and science marks:  95 99
'95 99'
>>> type(marks)
<class 'str'>
>>> marks=marks.split()
>>> type(marks)
<class 'list'>
>>> marks
['90', '97']
```

THANK YOU

Prof. Gayathri R S

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Prof. Gayathri R S
Computer Science and Engineering

Operator Precedence

- Determines the order of evaluation.
- Each programming language has its own rules for the order that operators are applied, called operator precedence
- Consider the following expression:

4+3*5

There are two possible ways in which it can be evaluated

4 + 3 * 5 → 4 + 15 → **19**

~~4 + 3 * 5 → 7 * 5~~ → **35**

Operator Precedence

- Operator precedence guarantees a consistent interpretation of expressions

  ***** has higher precedence than **+**. Therefore the expression will be evaluated as follows.

  4 + 3 * 5 → 4 + 15 → **19**

- If the addition is to be performed first, parentheses would be needed,

  (4 + 3) * 5 → 7 * 5 → **35**

- It is good programming practice to use parentheses even when not needed

Operator Associativity

Operator associativity defines the order that it and other operators with the same level of precedence are evaluated.

For example,

the associativity of exponentiation operator is right to left.

Therefore, 23**2** will be evaluated as follows

2 ** (3 ** 2) → **512**

~~(2 ** 3) ** 2 → **64**~~

Operator Precedence and Associativity

- The following table summarizes the operator precedence in Python, from highest precedence to lowest precedence. Operators in the same box have the same precedence.

- **Operators in the same box group left to right (except for exponentiation, which groups from right to left**).

- Note that comparisons, membership tests, and identity tests, all have the same precedence and have a left-to-right chaining feature.

- To see the complete documentation of operator precedence execute the below statement in Python.

```
print(help('>'))
```

Operator Precedence

Operator	Description
(expressions...)	Parenthesized expression
**	Exponentiation
"+x", "-x", "~x"	Positive, negative, bitwise NOT
"*", "/", "//", "%"	Multiplication, division, floor division, remainder
"+", "-"	Addition and subtraction
"<<", ">>"	Shifts
"&"	Bitwise AND
"^"	Bitwise XOR
"|"	Bitwise OR
"in", "not in", "is", "is not", "<", "<=", ">", ">=", "!=", "=="	Comparisons, including membership tests and identity tests
"not x"	Boolean NOT
"and"	Boolean AND
"or"	Boolean OR

THANK YOU

Prof. Gayathri R S
gayathrirs@pes.edu
Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Prof. Gayathri R S
Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Operator Polymorphism

Operator polymorphism

Operator polymorphism is a type of polymorphism where an operator behaves differently based on the type of the operands.

Operator polymorphism

For example:

+ acts as addition operator if the operands are numbers.

+ acts as concatenation operator if the operand are strings.

Note: Boolean literal True represents value 1 and False represents value 0

```
>>> 4+5.0
9.0
>>> True+True
2
>>> '4'+'5.0'
'45.0'
```

Operator polymorphism

- * Works as multiplication operator if the operands are numbers.
- * Works as repetition operator if one operand is a string and other operand a non- negative integer

```
>>> 5*3.0
15.0
>>> True*3
3
>>> '5'*3
'555'
```

THANK YOU

gayathrirs@pes.edu
Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Chitra G M, Gayathri R S

Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Control Structures

What is a Control Structure?

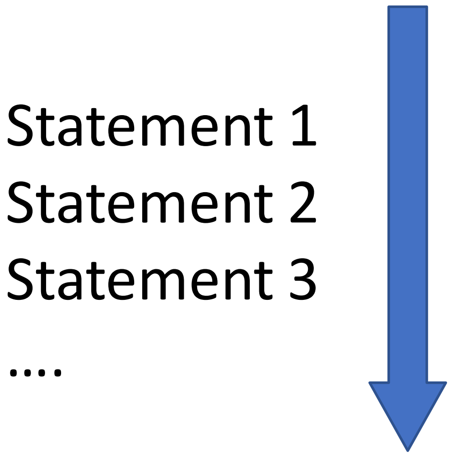
- C*ontrol flow* is the order that instructions are executed in a program.
- A **control statement** is a statement that determines control flow of a set of instructions.

There are three fundamental forms of control that programming languages provide,

- **sequential control**
- **selection control**
- **iterative control**

Sequential Control

Sequential statements are a set of statements whose execution process happens in a sequence.



Statement 1
Statement 2
Statement 3
….

Selection Control

- The selection statements are also known as Decision control statements or branching statements.

- The selection statement allows a program to test several conditions and execute instructions based on the Truth value of the condition

- Below are some of the decision control statements

  if

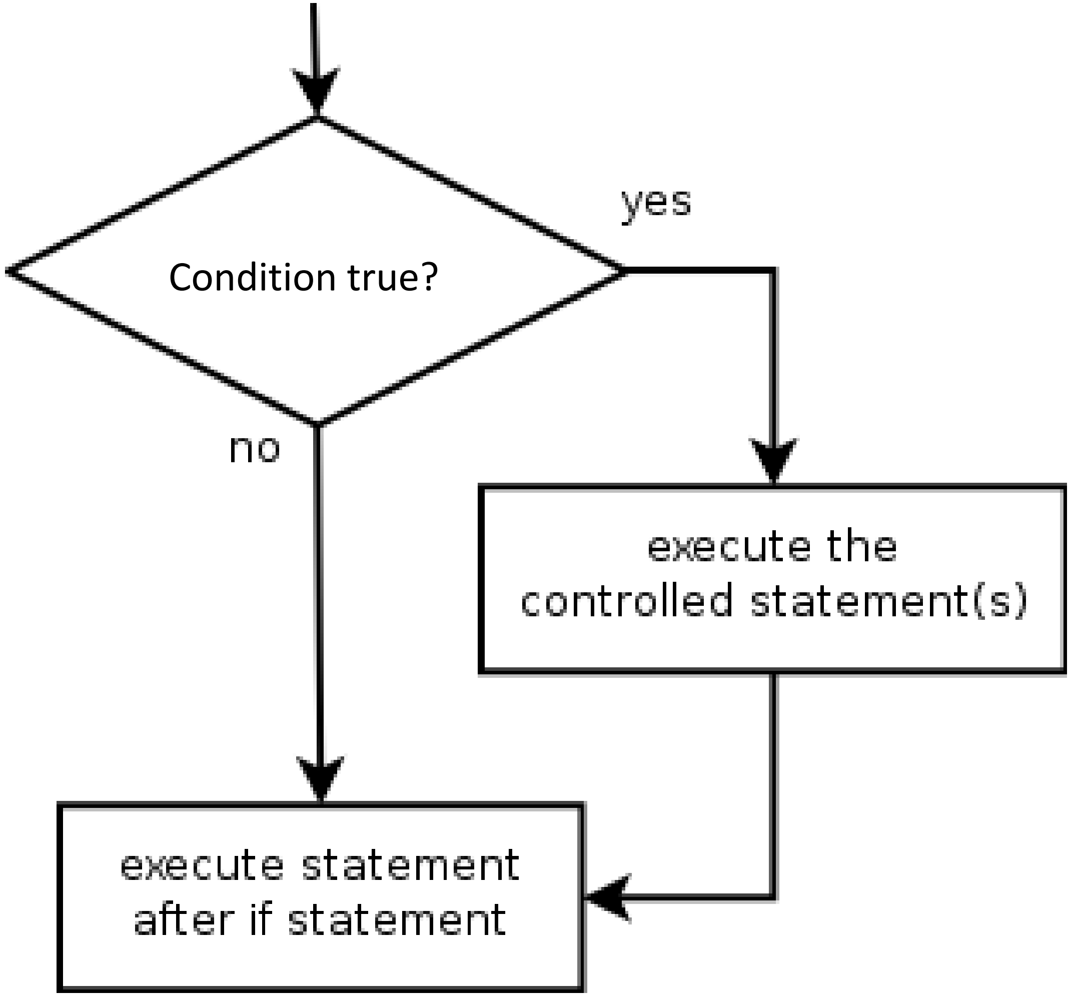
  if-else

  if-elif-else

Selection Control

simple if statement

Control structure runs a particular block of code only if certain condition is met.


Condition true?
yes
no
execute the controlled statement(s)
execute statement after if statement

Selection Control

simple if statement-general syntax

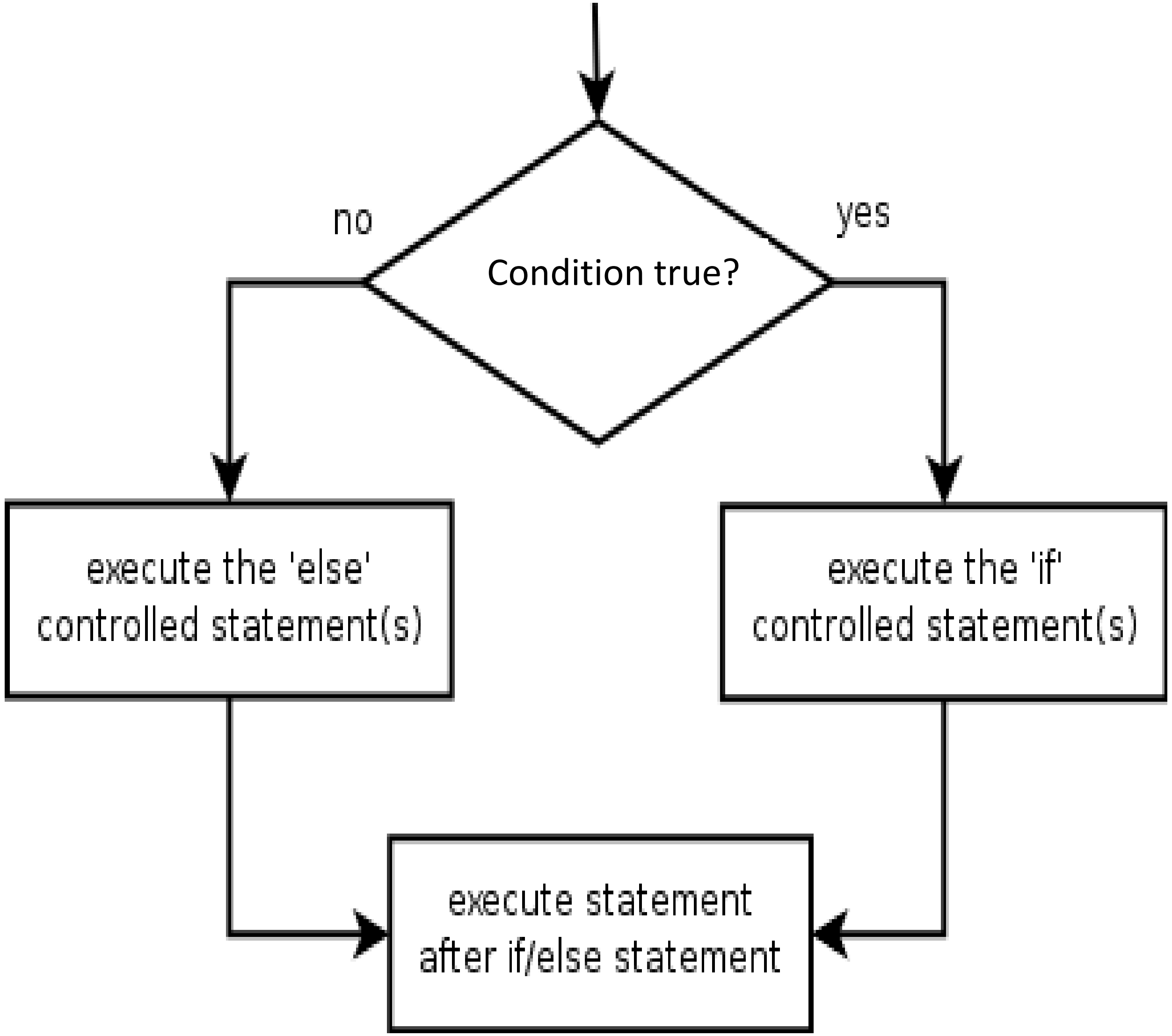
```
if <expression>:
        <block of code>
<statements after if statement>
```

- Expression can be any expression that evaluates to a Truth value
- Truth value means True or False
- Non zero number, non empty string, non empty list, True, etc are evaluated as True
- Number zero, None, False , empty string, empty list, etc are evaluated as False

Selection Control

if-else statement

A control structure that executes one block of statements if a certain condition is True, and a second block of statements if condition is False.


Condition true?
no
yes
execute the 'else' controlled statement(s)
execute the 'if' controlled statement(s)
execute statement after if/else statement

Selection Control

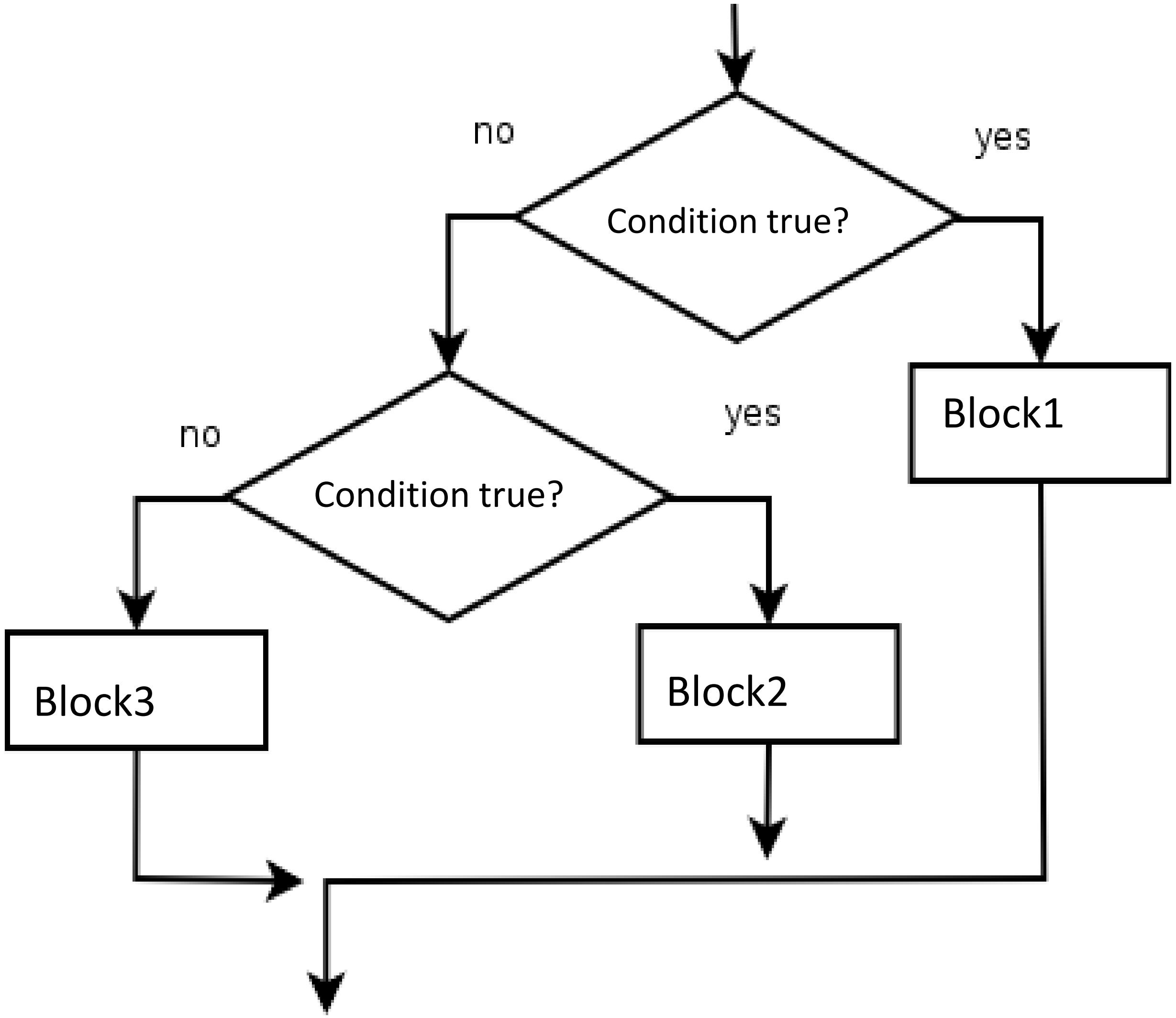
if-else statement-general syntax

```
if expression:
            < block of code>
else:
           <block of code>
< statements after the control structure>
```

Selection Control

if-elif-else

Any one block of the code will be executed based on the Truth value of the expression



Selection Control

if-elif-else general syntax

```
if <expression>:
            < block of code>
elif < expression>:
            < block of code>
else:
            < block of code>
<statements after the control structure>
```

Selection Control

if-elif-else

- Any number of elif constructs can be written after if statement.
- else construct is optional.
- Any one block of the code will be executed in the if elif else ladder based on the Truth value of the expression.

Selection Control

Indentation in Python

- One fairly unique aspect of Python is that the amount of indentation of each program line is significant.

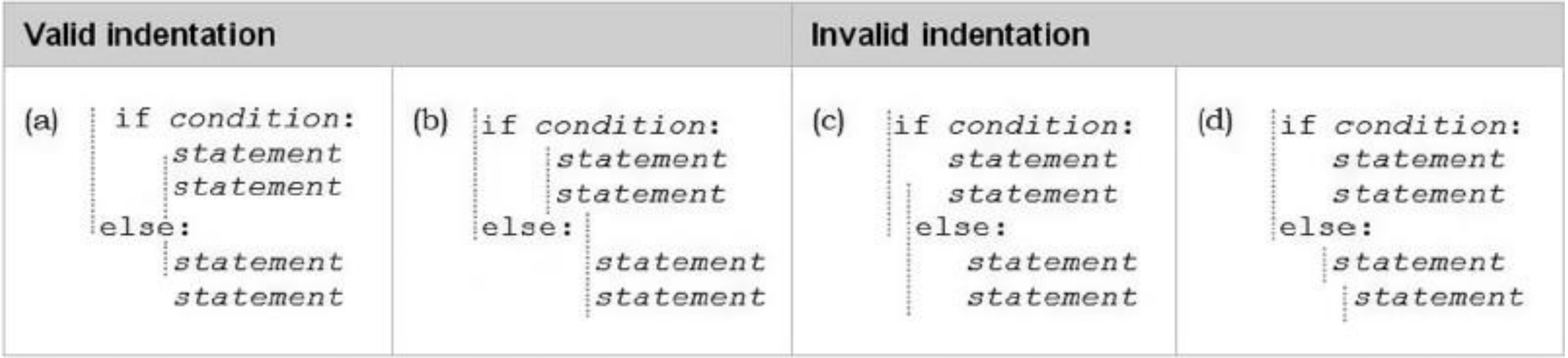
- In most programming languages, indentation has no affect on program logic—it is simply used to align program lines to aid readability.

- In Python, however, indentation is used to associate and group statements.

Selection Control

Valid and invalid indentation

A **compound statement** in Python may consist of one or more clauses. While four spaces is commonly used for each level of indentation, any number of spaces may be used, as shown below.



Valid indentation

(a)

```
 if condition:
    statement
    statement
else:
    statement
    statement
```

(b)

```
if condition:
    statement
    statement
else:
      statement
      statement
```

Invalid indentation

(c)

```
if condition:
   statement
   statement
 else:
    statement
    statement
```

(d)

```
if condition:
   statement
   statement
else:
   statement
    statement
```

Control Structures

- The set of statements following a header in Python is called a **suite** (commonly called a **block**).

- The statements of a given suite **must** all be indented the same amount.

- A header and its associated suite are together referred to as a **clause**.

Iterative Control

An **iterative control statement** is a control statement providing the repeated execution of a set of instructions.

An **iterative control structure** is a set of instructions and the iterative control statement(s) controlling their execution.

Because of their repeated execution, iterative control structures are commonly referred to as “loops”.

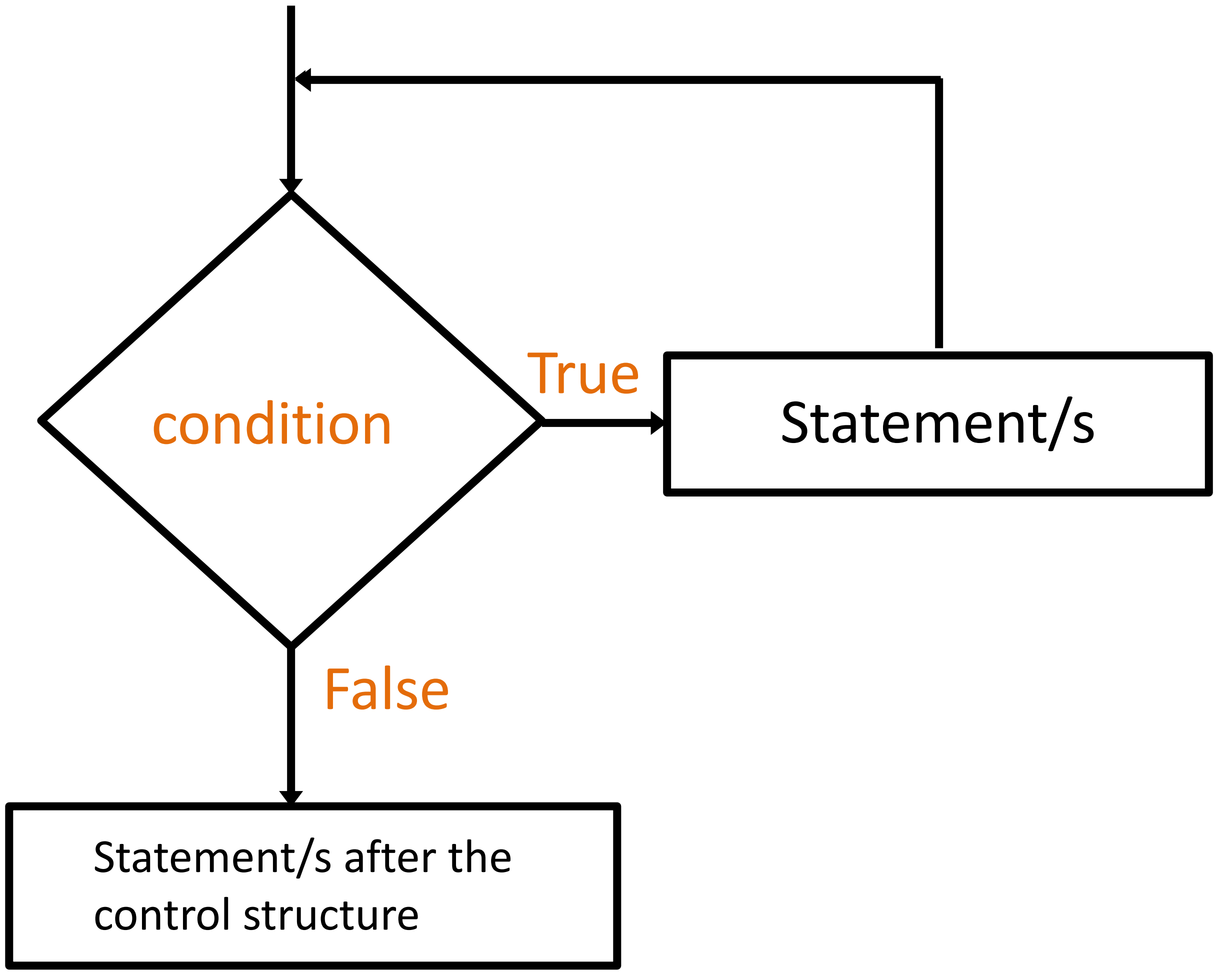
Iterative control- **while loop**

A **while statement** is an iterative control statement that repeatedly executes a set of statements based on the provided expression (condition).

There are 3 ingredients to any iterative statement:

1. Initial value of the iterative counter
2. Iterative condition
3. Updating the iterative counter

Iterative control- while loop



Iterative control- while loop

As long as the condition of a while statement is true, the statements within the loop are (re)executed

Once the condition becomes false, the iteration terminates and control continues with the first statement after the while loop

the condition may be false the first time a loop is reached and therefore the loop would never be executed.

Iterative control- while loop

Syntax:

```
while <condition>:
        <suite>
```

Example:

```
i=0
while i<5:#header or leader
        print("Hello ")
        i+=1
```

Iterative control- while loop

Infinite Loops

- An **infinite loop** is an iterative control structure that never terminates (or eventually terminates with a system error).
- Infinite loops are generally the result of programming errors.
- For example, if the condition of a while loop can never be false, an infinite loop will result when executed.

range()

- range is a built-in function
- range is lazy.
- range conceptually generates an arithmetic progression.

range()

range(start , stop , step)

- It returns a range object that produces a sequence of integers from start(inclusive) to stop(exclusive) by step.

  For example, range(0,4) produces 0,1,2,3
- range(i, j) produces i, i+1, i+2, ..., j-1.
- When step is given, it specifies the increment (or decrement).
- It is a Lazy function - The values come into existence only when we explicitly ask for it.

Why range() is needed in for loop?

- range function returns an iterable object that can be used in a for loop.
- range conceptually generates an arithmetic progression. The values given by range in a for loop can by used as indices for any sequence type.

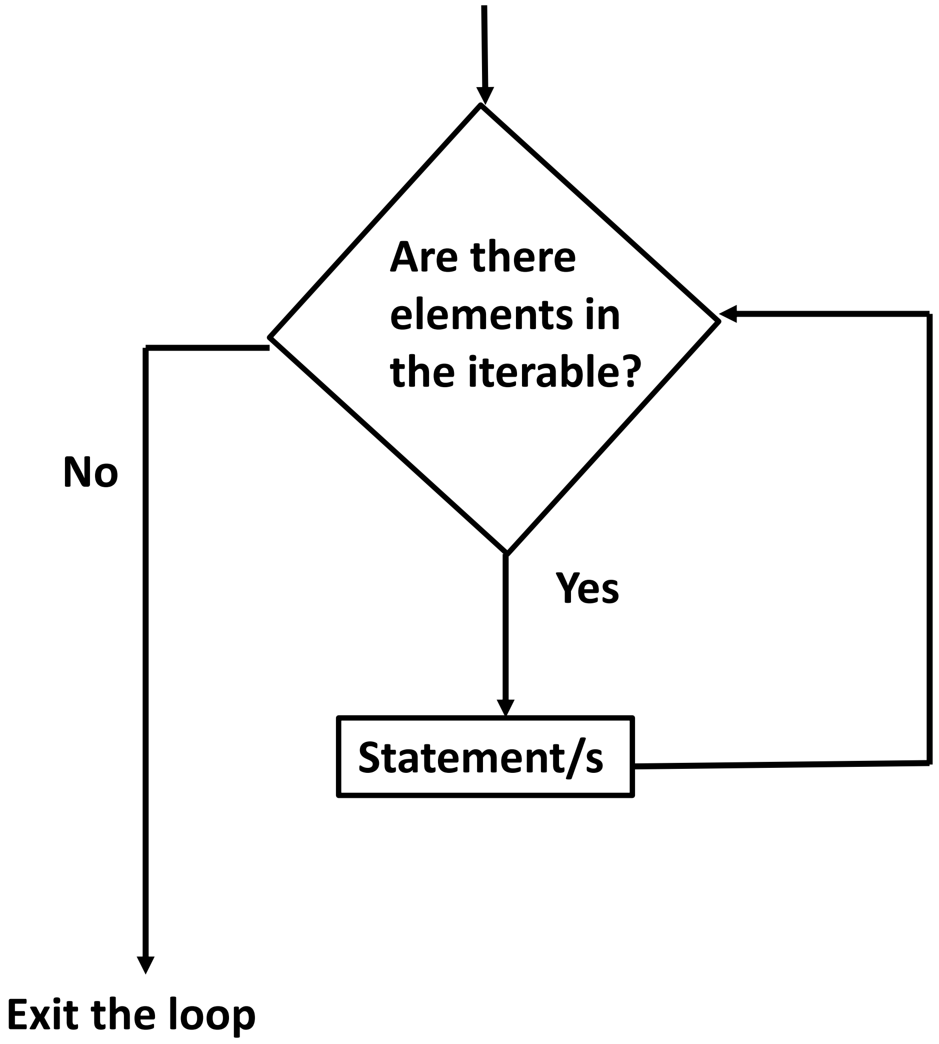
range()

```
>>> range(3)#lazy function
range(0, 3)
>>> list(range(3)) # start defaults to 0, and stop is omitted!
[0, 1, 2]
>>> list(range(3,6))
[3, 4, 5]
>>> list(range(3,3)) #does not give any element
[]
>>> list(range(-6,7,4))
[-6, -2, 2, 6]
>>> list(range(-16,-32,-4))
[-16, -20, -24, -28]
```

Iterative control- for loop

- Python has a statement that works exclusively on collections.
- examines each element and performs any action set by the programmer until there are no more elements left in the collection.
- for statement is a looping structure
- The number of times we execute the body or the suite is normally determinable

Iterative control- for loop



Are there elements in the iterable?

No

Yes

Statement/s

Exit the loop

Iterative control-for loop

Syntax:

```
for <target_variable> in <iterable>:
        <suite/body of for>
```

- An iterable is a collection of elements physically or conceptually.
- It has a built-in mechanism to give an element each time we ask
- It has a built-in mechanism to signal when there are no more elements.

Example:

```
for i in range(5): #header or leader
        print(i)
```

Semantics of *for* statement

1. start the iteration(walking through) of the iterable.
2. get the element to the variable.
3. execute the suite or the body.
4. repeat steps 2 and 3 until the iterable signals that it has no more elements.
5. move to the next statement and exit the for loop.

THANK YOU

Chitra G M, Gayathri R S

Department of Computer Science and Engineering