

0. Python Introduction

– By Srikanth Piniseti (Data Engineer @ SquadronData Inc)

Important terms:

- Bit – 0 or 1
- 1 Nibble – 4 Bits
- 1 Byte – 8 Bits
- 1 Word – 16 Bits
- Double Word – 32 bits
- Multiple Word – 64 bits or 128 bits

What is a Program ?

Set of Instructions are written in a SPECIFIC SEQUENCE or FORMATE in a computer to accomplish or perform a given task

- Example:
- Adding of two numbers
- Analyzing changes in the weather (Import the weather library and do some predictions - Data Science)
- Taking input of a users and organizing them into a excel file (Use of input function)

Types of Softwares:

Type of Software	Description	Examples
System Software	Software that manages computer hardware and provides a platform for running application software.	Operating Systems: Windows, macOS, Linux; Device Drivers; Utility Software: Antivirus programs, disk cleaners.
Application Software	Software designed for specific tasks or applications.	Productivity: Microsoft Word, Google Docs; Communication: WhatsApp, Gmail; Multimedia: Adobe Photoshop, Spotify; Entertainment: Minecraft, Candy Crush; Educational: Khan Academy, Duolingo; Business: QuickBooks, Salesforce; Web Browsers: Google Chrome, Mozilla Firefox.
Embedded Software	Software embedded within hardware devices.	Firmware in consumer electronics, automotive systems, medical devices.

Type of Software	Description	Examples
Open Source Software	Software with source code freely available to the public.	Linux operating system, Apache web server, Firefox web browser.
Proprietary Software	Software owned by a specific company with restricted access to the source code.	Microsoft Office, Adobe Creative Suite.
Freeware and Shareware	Freeware: Software distributed for free; Shareware: Software with trial versions or limited features.	Freeware: 7-Zip, Audacity; Shareware: WinRAR, IDM (Internet Download Manager).
Cloud-Based Software	Software accessed over the internet, typically through a web browser.	Google Workspace, Microsoft Office 365, Dropbox.

Different types of computer languages ?

Computer languages can be categorized based on different criteria, such as their level of abstraction, programming paradigm, and execution model. Here's an organized overview of the different types of computer languages with examples:

1. Level of Abstraction:

- **Low-level languages:** These languages are close to the machine-level instructions and provide direct access to the computer's hardware. Examples: Assembly language, machine code.
 - Assembly language: `mov eax, 5` (move the value 5 into the eax register)
 - Machine code: `1011000100100000` (binary representation of a machine instruction)
- **Middle-level languages:** These languages are somewhat abstracted from the hardware and provide a higher level of abstraction than low-level languages. Example: C.
- **High-level languages:** These languages are highly abstracted from the hardware and are closer to human-readable form. Examples: Python, Java, C++, C#, JavaScript, Ruby.

2. Programming Paradigms/Approaches:

- **Procedural languages:** These languages follow a step-by-step approach and use procedures or routines to perform tasks. Examples: C, Pascal, FORTRAN.
- **Object-Oriented Programming (OOP) languages:** These languages are based on the concept of objects, which encapsulate data and behavior. Examples: Java, C++, C#, Python, Ruby, Objective-C.
- **Functional languages:** These languages treat computation as the evaluation of mathematical functions and avoid changing state and mutable data. Examples: Lisp, Haskell, Erlang, Clojure, Scala.
- **Declarative languages:** These languages specify the desired result without explicitly describing the control flow. Examples: SQL (for databases), HTML, XML.

- **Scripting languages:** These languages are designed for writing scripts that can control and automate the execution of tasks. Examples: Python, Bash (Unix shell scripting), JavaScript, Perl, Ruby.

3. Execution Model:

- **Compiled languages:** These languages are translated into machine code by a compiler before execution. Examples: C, C++, Rust, Go, Swift.
- **Interpreted languages:** These languages are executed directly by an interpreter without prior compilation to machine code. Examples: Python, JavaScript, Ruby, Perl, Bash.

4. Domain-Specific Languages (DSLs):

These languages are designed for specific domains or applications and offer expressive power tailored for that particular domain. Examples:

- **HTML:** A markup language for creating web pages.
- **CSS:** A style sheet language used for describing the presentation of web pages.
- **SQL:** A language for managing and querying relational databases.
- **LaTeX:** A markup language for typesetting documents, especially technical or scientific documents.
- **R:** A language and environment for statistical computing and graphics.
- **MATLAB:** A language for numerical computing, data analysis, and visualization.

It's important to note that some languages can fall into multiple categories, as they may incorporate features from different paradigms or have different execution models depending on the implementation.

Additionally, there are other classification criteria for programming languages, such as:

- **Strongly typed vs. Weakly typed:** Languages that enforce strict type checking (e.g., Java, C++) vs. languages with more flexible type systems (e.g., Python, JavaScript).
- **Static vs. Dynamic typing:** Languages where variable types are determined at compile-time (e.g., Java, C++) vs. languages where variable types are determined at runtime (e.g., Python, JavaScript).
- **Imperative vs. Declarative:** Languages that focus on describing how to perform a task (e.g., C, Java) vs. languages that focus on describing what the desired result is (e.g., SQL, HTML).

The choice of programming language often depends on the specific requirements of the project, the domain or application area, the available tools and libraries, the performance and scalability needs, and the personal preferences and expertise of the development team.

What is runtime and compile time error ?

- **Compile-Time Error:**
 - Occurs during compilation, detected by the compiler, and prevents the program from being executed.
 - When you try to compile a program that contains errors, the compiler will report those errors in the console output, and the program will not be successfully compiled into an executable file.
- **Runtime Error:** Occurs during program execution, detected while the program is running, and can lead to program crashes or unexpected behavior.

What is an Algorithm ?

An algorithm is a step-by-step procedure or set of instructions designed to solve a specific problem or perform a particular task. Algorithms are fundamental to computer science and programming, as they provide a systematic way to solve problems efficiently.

- **Types:**

- **Pseudo-code** : represents the algorithm in a way that is in between a programming language and English statements

Food_Item, Quantity, Unit_Price, Total_Cost are variables used in the pseudo code.

Input Food_Item, Quantity

Unit_Price = 10

Total_Cost = Unit_Price * Quantity






To assign a value to the variable, we can use the "=" symbol. It is called as assignment operator.

Display "Order successfully placed for ", Food_Item

Display Total_Cost

Variables are like containers for data (i.e., they hold the data) and the value of the variable can vary.

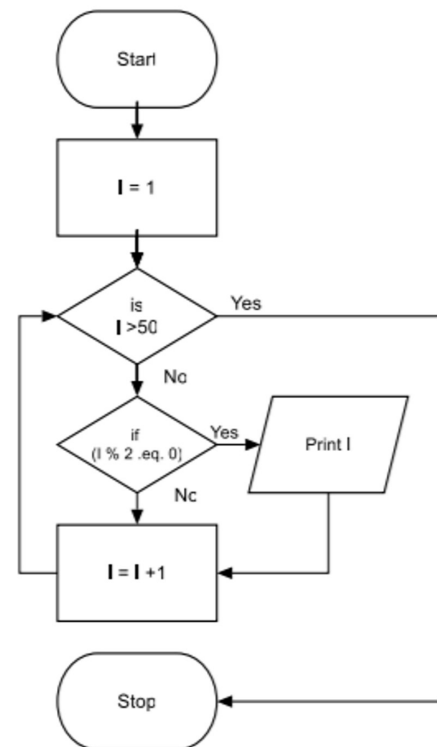
- Note: Here, the assumptions are:
 1. The customer buys only 1 food item at a time.
 2. The price of 1 unit of any food item is \$10.
- **Flowchart** : represents the algorithm in a diagrammatic way

Symbol	Usage	Description
	arrowhead	represents the direction of flow
	terminal	represents the start and end of a program
	process	represents an action, process or operation
	decision	indicates a question to be answered (yes/no or true/false questions). The flowchart path can split into various branches depending on the answer
	input/output	represents the input and output of data

Algorithm & Flowchart to find Even number between 1 to 50

Algorithm

- Step-1 Start
- Step-2 $I = 1$
- Step-3 IF ($I > 50$) THEN
GO TO Step-7
ENDIF
- Step-4 IF ($(I \% 2) = 0$) THEN
Display I
ENDIF
- Step-5 $I = I + 1$
- Step-6 GO TO Step-3
- Step-7 Stop



Here are different types of algorithms along with examples for each type:

1. Sequential Algorithms:

- These algorithms follow a linear sequence of steps, where each step is executed one after the other.
- Example: Linear Search Algorithm

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

arr = [4, 2, 7, 1, 9, 5]
target = 7
print(linear_search(arr, target)) # Output: 2 (index of 7 in arr)
```

2. Divide and Conquer Algorithms:

- These algorithms divide the problem into smaller subproblems, solve them recursively, and then combine the solutions.
- Example: Merge Sort Algorithm

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid])
```

```

    right_half = merge_sort(arr[mid:])
    return merge(left_half, right_half)

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

arr = [4, 2, 7, 1, 9, 5]
print(merge_sort(arr)) # Output: [1, 2, 4, 5, 7, 9]

```

3. Greedy Algorithms:

- These algorithms make a series of choices, each of which is the best or optimal choice at the moment.
- Example: Dijkstra's Shortest Path Algorithm

```

def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    unvisited = set(graph.keys())
    while unvisited:
        min_node = min(unvisited, key=distances.get)
        unvisited.remove(min_node)
        for neighbor, weight in graph[min_node].items():
            new_distance = distances[min_node] + weight
            if new_distance < distances[neighbor]:
                distances[neighbor] = new_distance
    return distances

graph = {
    'A': {'B': 3, 'C': 2},
    'B': {'C': 1, 'D': 5},
    'C': {'D': 2},
    'D': {}
}
print(dijkstra(graph, 'A')) # Output: {'A': 0, 'B': 3, 'C': 2, 'D': 4}

```

4. Dynamic Programming Algorithms:

- These algorithms break down complex problems into simpler overlapping subproblems and solve each subproblem only once.

- Example: Fibonacci Sequence using Dynamic Programming

```
def fibonacci(n):
    fib = [0, 1]
    for i in range(2, n + 1):
        fib.append(fib[i - 1] + fib[i - 2])
    return fib[n]

print(fibonacci(6)) # Output: 8 (fibonacci(6) = 8)
```

5. Backtracking Algorithms:

- These algorithms solve problems incrementally by building candidates for the solution and abandoning a candidate as soon as it determines that it cannot be completed.
- Example: N-Queens Problem using Backtracking

```python

```
def is_safe(board, row, col):
 for i in range(col):
 if board[row][i] == 1:
 return False
 for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
 if board[i][j] == 1:
 return False
 for i, j in zip(range(row, len(board), 1), range(col, -1, -1)):
 if board[i][j] == 1:
 return False
 return True
```

```
def solve_n_queens(board, col):
 if col >= len(board):
 return True
 for i in range(len(board)):
 if is_safe(board, i, col):
 board[i][col] = 1
 if solve_n_queens(board, col + 1):
 return True
 board[i][col] = 0
 return False

def n_queens(n):
 board = [[0] * n for _ in range(n)]
 if not solve_n_queens(board, 0):
 return False
 return board

def print_board(board):
 for row in board:
```

```
print(row)

print_board(n_queens(4))
...
```

These are some examples of common types of algorithms, each with its own approach to problem-solving and implementation details.

## \*What are control Statements?\*

Control statements are programming constructs that alter the flow of execution in a program based on certain conditions or criteria. They allow programmers to control the order in which statements are executed and to make decisions about which code blocks to execute. Common control statements include:

### 1. Conditional Statements:

- Conditional statements allow the execution of different code blocks based on whether a specified condition evaluates to true or false.
- Examples:
  - **if statement:** Executes a block of code if a specified condition is true.
  - **if-else statement:** Executes one block of code if a condition is true and another block if it is false.
  - **if-elif-else statement:** Executes one of several blocks of code based on multiple conditions.

Example:

```
x = 10
if x > 0:
 print("x is positive")
elif x < 0:
 print("x is negative")
else:
 print("x is zero")
```

### 2. Looping Statements:

- Looping statements allow the execution of a block of code repeatedly based on certain conditions.
- Examples:
  - **for loop:** Executes a block of code a specified number of times.
  - **while loop:** Executes a block of code as long as a specified condition is true.
  - **do-while loop** (available in some languages): Similar to a while loop, but the block of code is executed at least once before checking the condition.

Example:



```
Using a for loop
for i in range(5):
 print(i)

Using a while loop
x = 0
while x < 5:
 print(x)
 x += 1
```

### 3. Transfer Statements:

- Transfer statements change the normal flow of execution in a program.
- Examples:
  - **break statement:** Terminates the loop it is in and transfers control to the next statement outside the loop.
  - **continue statement:** Skips the current iteration of a loop and continues with the next iteration.
  - **return statement:** Exits a function and optionally returns a value.

Example:

```
for i in range(10):
 if i == 5:
 break
 print(i)
```

### 4. Exception Handling Statements:

- Exception handling statements allow the handling of runtime errors or exceptional situations gracefully.
- Examples:
  - **try-except statement:** Executes a block of code and handles any exceptions that occur within it.
  - **try-except-finally statement:** Executes a block of code, handles any exceptions, and executes a cleanup block of code regardless of whether an exception occurred.

Example:

```
try:
 x = 1 / 0
except ZeroDivisionError:
 print("Division by zero error occurred")
```

These control statements provide the flexibility to create complex programs by controlling the flow of execution based on various conditions and requirements.

## ***What is Platform Independent and Machine Independent?***

### **1. Platform Independent:**

- It means software can run on different types of operating systems like Windows, macOS, or Linux without needing to be changed.
- Examples: Java and Python programs can run on any operating system without modification.

### **2. Machine Independent:**

- It means software can run on different types of computer hardware like desktops, servers, or smartphones without needing to be changed.
- Examples: Java programs can run on any hardware architecture because they are compiled into bytecode, which is then interpreted by the Java Virtual Machine (JVM). Similarly, .NET languages like C# can run on different hardware architectures because they compile to Common Intermediate Language (CIL), which is then executed by the .NET runtime.