Python's Indispensability in Artificial Intelligence and Machine Learning: A Review

[1]Fatoba, Toyosi Mercy; [2]Oyeyemi, Dare Azeez; [3]Oluyele, Sunday Anthony; & [4]Adeniyi, Victor Oluwatobi.
[1]Mechanical Engineering Department, The Federal Polytechnic, Ado-Ekiti;
[2]Computer Science Department, University of Ibadan;
[3]Computer Engineering Department, Federal University Oye Ekiti;
[4]Statistics Department, University of Lagos.

Corresponding Author's Name & Email Address
Fatoba, Toyosi Mercy, fatobatoyosimercy@gmail.com.

DOI: 10.56201/ijemt.vol.11.no4.2025.pg266.303

ABSTRACT
This review paper explores the vital role played by the programming language Python in Artificial Intelligence (AI) and Machine Learning (ML). It will discuss the origin of AI and ML, the first programming language used for it, and its bottlenecks will be made known. There will also be an exposition on the emergence of Python, its elements, and the comparison between Python and LISP. The adoption and integration of Python into AI and ML will be considered, and the advanced components of Python and their uses in AI and ML will also be discussed. The trajectory of AI and ML with Python and the game-changing benefits of using Python in AI and ML will be assessed. Suggestions will be given on what should be improved in Python to improve AI and ML.

Keywords*:* AI; ML; Python; Automation; Data analytics.

1.0. INTRODUCTION

1.1 The Inception of Artificial Intelligence (AI) and Machine Learning (ML).

Although artificial intelligence and machine learning appear to some as new concepts, they have been in existence for over six (6) decades. Also, most people usually use the terms AI and ML interchangeably, whereas they are different entities with integrative features. These technologies have brought about tremendous changes in society and positively affected businesses and lives (Toyosi et. al., 2024).

Artificial intelligence (AI) originated in the 1940s, inspired by Isaac Asimov's Three Laws of Robotics and influenced by Alan Turing's pivotal contributions to code-breaking machines during World War II. The term "AI" was formally introduced in 1956 during the Dartmouth Summer Research Project on AI, which marked the beginning of dedicated AI research efforts aimed at developing machines with the capacity to mimic and replicate human intelligence. (Haenlein & Kaplan, 2019). Artificial Intelligence (AI) refers to the ability of computer systems to perform tasks that typically necessitate human intelligence and input. AI systems are created to continually acquire knowledge and advance their capabilities, rendering them an important asset across various industries (Pranav & Sarma, 2023).

Machine learning helps in the development of systems that use available data to learn and bring about performance improvement on the tasks assigned. Machine learning is an integral concept of artificial intelligence that designs models through its algorithm and utilizes training data to predict or decide. ML can be used in diverse ways, which includes helping to simplify human tasks, so it serves as a valuable tool in carrying out human endeavors. The name 'Machine learning' came into existence in the year 1959, through the erudite scientist, named Arthur Samuel, who was a trailblazer in the area of computer gaming and artificial intelligence and also a staff of IBM at that time. Several machine-learning inventions came afterward that were trained repeatedly by humans. (Pranav & Sarma, 2023).

1.2. The Relationship between Artificial Intelligence (AI) and Machine Learning (ML)

AI and ML are relative, even though they are not the same, and it would be easier to comprehend their relationship when they are compared. AI encompasses a wider conceptual range, which gives a system or machine the ability to think, detect, do, and make changes in a human-like manner. ML serves as an aspect of AI that is applied to machinery, which enables them to utilize collected data as a knowledgeable guide to learn independently. A typical way to show the relationship between AI and ML is knowing that AI is a wide-reaching name that entails vast algorithmic procedures, and ML is dependent on it to function optimally like every one of its aspects such as robotics, expert systems, deep learning, and natural language processing, amongst others (Oracle, 2022).

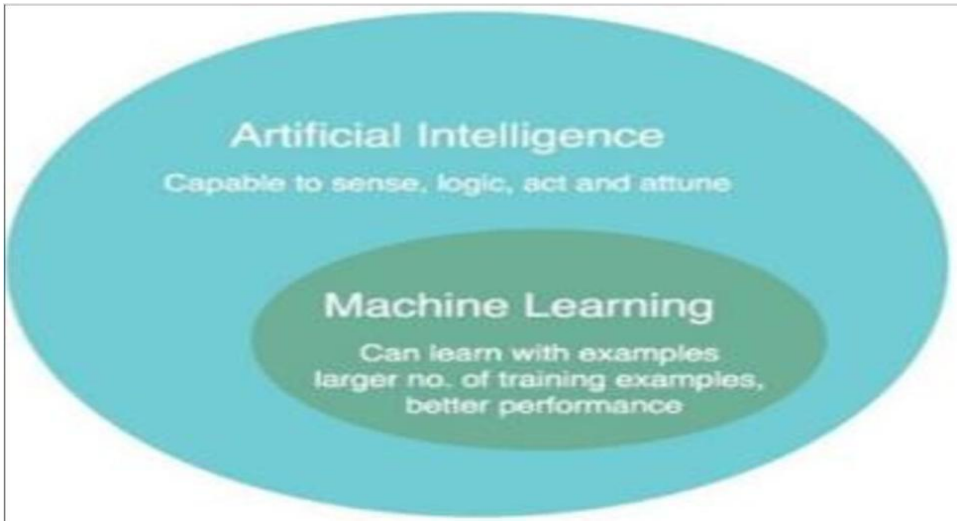

Fig. 1: *Machine Learning: A Subset of Artificial Intelligence (Ogunsanya & Taiwo, 2024)*

1.3 The First Programming Language Used in Artificial Intelligence (AI) and Machine Learning (ML)

List Processing (LISP) was developed in 1958 by John McCarthy as the first practical and functional programming language for Artificial Intelligence. It is a program that relies on lambda's idea and functional mathematical theories. LISP has been used to write a lot of useful AI applications (Neumann, 2002).

LISP serves as a useful AI research programming language, and many ideas for computer science, such as the read-eval-print loop, tree data structures, conditionals, higher-order functions, the self-hosting compiler, dynamic typing, and automatic storage management, were built on it. The major data structure of LISP is the 'Linked List,' and it's the source of its code (Adetiba et al., 2021).

1.4 The Major Bottleneck of LISP

The bottleneck of List Processing (LISP) is that it was built on mathematical notation, which uses lambda calculus, a concept that is not generally available in institutions of higher learning for its operations. This factor made it less comprehensible or usable to elementary programmers, unlike some other Object-oriented languages such as Python, amongst others; it's also not easy to master for experienced programmers, which implies no user-friendliness (Adetiba et al., 2021).

2.0 BODY

2.1 The Discovery of the Python Programming Language.

The multifunctional programming language called Python was formulated at the Centrum Wiskunde and Informatica (CWI), Netherlands, by Guido van Rossum in 1980 and was officially launched with the version named Python 0.9.0. in 1991 (Rossum, 2009; Venners, 2003).

Python came into existence as a better substitute to the already existing ABC programming language, and it also bridged the gap of the ABC programming language with its capability to handle exceptions and its ability to interact with the Amoeba's operating system (Van Rossum, 2000). Improvement was made to the, previously launched Python programs with the introduction of the Python 2.0 version in the year 2000, which comprises more interesting attributes such as the reference counting tool, list comprehension, garbage collection, Unicode support, and cycle-detecting (Kuchling & Zadka, 2000).

Several versions of the Python program have been created after version 2.0. However, python version 3.9 has remained the oldest version that is supported security-wise because version 3.8 got to its end-of-life era. Python version 3.13 is the most recent Python program released; also, the only active versions of the Python program are versions 3.13 and 3.12 (Status of Python Versions, 2024).

Python, being a top-notch and all-around useful programming language, is generally acceptable for its readability through its usage of indentations in its algorithm (Kuhlman, 2011). It is a dynamic programming language in that it gives room for multi-programming patterns, which comprise functional, structured, and object-oriented programming. The comprehensiveness and extensiveness of its standard library are the reason why it's also referred to as a ‘batteries included’ language (Hulatt & Frietas, 2024; PEP 206 – Python Advanced Library, 2000).

2.2 The Basic Elements of Python and their Purpose

Python programming language has diverse basic elements that form the foundation upon which every advanced algorithm progression is built, and they are explicitly unfolded in this section.

2.2.1 Variable: is the name used to assign value or store data in Python programming. Only letters, digits, and underscore can be used as a variable name, it's case sensitive, and Python programming

language keywords (such as else, if, for, print...) can't be used as variable names (Geeksforgeeks, 2025).

Table 1.
Variable.

S/N	Variable	Operators	Value/Data	Syntax/ Example
1.	X	=	5	X = 5
2.	Her_Size	=	21.5	Her_Size = 21.5
3.	Cycle	=	Four	Cycle = “Four”

2.2.2 Operators: special characters and symbols used to manipulate data and carry out diverse tasks in Python are called Operators. There are various types of operators in Python, including arithmetic, assignment, comparison, logical, bitwise, and membership operators (Singh, 2024).

- **Arithmetic operators:** are used to carry out operations according to the mathematical order, such as addition, subtraction, multiplication, division, exponentiation, modulus, and floor division (Geeksforgeeks, 2025).

Table 2.
Arithmetic Operators.

S/N	Arithmetic Operators	Symbol	Syntax/ Example
1.	Addition	+	X = 2 Y = 1 X + Y
2.	Subtraction	-	X - Y
3.	Division	/	X / Y
4.	Multiplication	*	X * Y
5.	Exponentiation	**	X ** Y
6.	Modulus	%	X % Y
7.	Floor Division	//	X // Y

- **Assignment Operators**: are used for attaching or assigning values to variables, there are diverse types of assignment operators as listed below.

Table 3.
Assignment Operators (Singh, 2024).

S/N	Assignment Operators	Symbol	Syntax/ Example	Definition
1.	Simple Assignment Operators	=	X = 2	For assigning a value to the variable.
2.	Addition Assignment Operators	+=	X += 2	For adding a value to the variable and assigning the result to the variable.
3.	Subtraction Assignment Operators	-=	X -= 2	For subtracting a value from the variable and assigning the result to the variable.
4.	Division Assignment Operators	/=	X /= 2	For dividing the variable by a value and assigning the result to the variable.
5.	Multiplication Assignment Operators	*=	X *= 2	For multiplying a variable by a value and assigning the result to the variable.
6.	Exponentiation Assignment Operators	**=	X **= 2	For raising the variable to the power of a value and assigning the result to the variable.
7.	Floor Division Assignment Operators	Python	X //= 2	For dividing the variable by a value and returning the largest integer either less or equal to the result.
8.	Modulus Assignment Operators	%=	X %= 2	For finding the remainder when the variable is divided by a value and assigning the result to the variable.

- **Comparison Operators:** are used to compare values and they always result in either True or False based on the conditions. The types of comparison operators are listed in the figure below.

Table 4.
Comparison Operators (Geeksforgeeks, 2025).

S/N	Comparison Operators	Symbol	Syntax/ Example	Definition
1.	Greater Than	>	X = 2 Y = 1 X > Y	True: if the left operand is greater than the right.
2.	Less Than	<	X < Y	True: if the left operand is lesser than the right.
3.	Equal To	==	X == Y	True: if both operands are equal.
4.	Not Equal To	!=	X!= Y	True: if both operands are not equal.
5.	Greater Than or Equal To	>=	X >= Y	True: if the left operand is greater than or equal to the right.
6.	Less Than or Equal To	<=	X <= Y	True: if the left operand is lesser than or equal to the right.

- **Logical Operators**: are used to merge conditional statements in terms of True or False. The logical operators are AND, OR, and NOT (Geeksforgeeks, 2025).

Table 5.
Logical Operators (Geeksforgeeks, 2025).

S/N	Logical Operators	Syntax	Definition
1.	AND	X and Y	If both operands are True, the output is True
2.	OR	X or Y	If one of the operands is True, the output is True.
3.	NOT	not Y	If the operand is False, the output is True.

- **Bitwise Operators**: are used to perform operations with integers bit by bit and they operate on binary numbers (Geeksforgeeks, 2025).

Table 6.
Bitwise Operators (Singh, 2024).

S/N	Bitwise Operators	Symbol	Syntax/ Example	Description
1.	Bitwise AND	&	X & Y	It carries out a bitwise AND operation on the binary representations of two integers.
2.	Bitwise OR	|	X | Y	It carries out a bitwise OR operation on the binary representation of two integers.
3.	Bitwise NOT	~	~X	It carries out a bitwise NOT operation on the binary representation of an integer.
4.	Bitwise XOR	^	X ^ Y	It performs a bitwise XOR operation on the binary representation of two integers.
5.	Bitwise Right Shift	>>	X>>	It moves the bits of the left operand to the right by the number of positions indicated by the right operand.
6.	Bitwise Left Shift	<<	X<<	It shifts the bits of the left operand to the left by the exact number of positions specified by the right operand, ensuring precise data manipulation.

- **Membership Operators**: are used to assess whether a specific value or variable is present within a given sequence.

Table 7.
Membership Operators (Geeksforgeeks, 2025).

S/N	Membership Operators	Syntax/ Example	Description
1.	IN	Ages = [92, 93, 94, 95] print (93 in Ages) Result: True	It is used to check if a character, substring, or element exists in a sequence. It results in 'True' if the specified element is found; otherwise, it is 'False.'
2.	NOT IN	Ages = [92, 93, 94, 95] print (93 not in Ages) Result: False	It is used to check if a character, substring, or element doesn't exist in a sequence. It results in 'True' when the variable is absent from the specified sequence, while it results in 'False,' if the variable is found.

2.2.3 Data Types: In programming, data types are ways data are classified and they define the kind of value they represent and the operations that can be performed upon them (NTU Library, 2024).

- **Numerical Data Types:** Python provides some built-in data types to handle different kinds of numerical values which are important for mathematical computations as needed in AI/ML.

Table 8.
Numerical Data Types

S/N	Data Types	Syntax/ Example	Description
1.	Integers (int)	x = 5	This is used to represent whole numbers, positive or negative, without fractions or decimals (NTU Library, 2024)
2.	Floating-Point Numbers(float)	y = 3.14	This is used for real numbers with decimal points and used for precision calculations (Ali, 2015).
3.	Complex Numbers (complex)	z = 2 + 3j	This is used for numbers with real and imaginary parts - they are useful in scientific computing and engineering fields (Ali, 2015).

- **Sequence Data Types:** These are used to handle ordered collections of items. Examples are below:

Table 9.
Sequence Data Types

S/N	Data Types	Syntax/ Example	Description
1.	Strings (str)	s = "Hello"	They are used for sequences of characters; they are immutable, i.e., modifying them will create a new string object (Pozo Ramos, 2025).
2.	List (list)	l = [1, "a", 3.5]	These are ordered, mutable collections that can hold heterogeneous elements (Abhirami, 2024).
3.	Tuples (tuple)	t = (1, "b", 2.5)	These are ordered, immutable collections; they support heterogeneous elements (Severance, 2016).
4.	Dictionaries (dict)	d = {"key": "value"}	They are used for key-value pairs, mutable, and maintain insertion order from Python 3.7+ (Patnaik, 2023).
5.	Sets (set)	s = {1, 2, 3}	They are an unordered collection of unique, immutable-type elements; mutable as a whole (NTU Library, 2024).

- **Boolean Data Type**: This is used to represent logical values for truth conditions in programming

Table 10.
Boolean Data Type

Data Type	Syntax/ Example	Description
Booleans (bool)	flag = True	They typically are True or False and are used in controlling program flow with conditionals and logical operators (Ali, 2015).

2.2.4 Functions: They allow programmers to put an entire collection of statements into a block and this block can now be executed as many times as we want by calling its name (Simplilearn, 2025). It is implemented using the def keyword then the function name, a pair of parenthesis, and a colon (e.g., def sum(x, y):). The block of code that would contain the body of the function will be indented below the def line (Parlante, 2020).

Table 11.
Function Types

S/N	Function Types	Syntax/ Example	Description
1.	User-defined functions	def my_func():	These are written by programmers using the 'def' keyword to perform specific tasks according to a project's requirements (Singh, 2024).
2.	Lambda functions	lambda x: x + 1	These are small, anonymous functions defined with the 'lambda' keyword, they are limited to a single expression and are often used with map(), filter(), and sorted() (Rao, 2024; Singh, 2024).
3.	Recursive functions	def recur(): return recur()	These are functions that call themselves to solve problems using repeated subproblem breakdown, they always include a base case to terminate recursion (Rao, 2024; Singh, 2024).

2.2.5 Control Flow Statements: are fundamental constructs in programming that dictate the order in which instructions are executed. They enable dynamic decision-making and repetitive task automation, enhancing the flexibility and efficiency of code (StudySmarter, 2024). The two main types of control flow discussed in this section are the conditionals and loops.

- **Conditional Statements:** allow programs to execute different code blocks based on specified conditions defined by the programmer (Chen et al., 2025). Key constructs are explained in the table below.

Table 12
Conditional Statements

S/N	Conditional Statement Types	Description	Syntax/ Example
1.	If Statements	This is the simplest form of branching, where code executes only if a certain condition is true (Chen et al., 2025).	temperature = float(input("Enter the temperature in °C: ")) 1f temperature < 15: print("it's cold!") *#The output of this would depend on the value of temperature inputted by the user if less than 15.*

2.	If-else and elif:	This extends the decision-making to multiple outcomes (Sebesta, 2018).	score = float(input("Enter your score"))
			if score >=90:
			print("Your grade is A")
			if score >=65:
			print("Your grade is B")
			else:
			print("Your grade is C")

3.	Ternary Operators:	This is a concise one-line conditional statement (Chen et al., 2025). It is like a short "if-else" statement squeezed into one line.	if weather=="sunny": print ("Go to the beach") else: print("Stay home")

- **Loop Statements:** Repeat a block of code until a condition is met, reducing redundancy (StudySmarter, 2024). Loops are indispensable for tasks like data processing (e.g., iterating through datasets), simulations, and automating repetitive operations (e.g., batch file processing). Common types of loops are explained in Table 13.

Table 13
Types of Loop Statements

S/N	Types of Loop Statements	Description	Syntax/ Example
1.	For Loops	It iterates over sequences (e.g., lists, strings) (Gaddis, 2024).	for i in range(5): print(i) *# Outputs: 1 2 3 4 5*
2.	While Loops	This executes as long as a condition remains true.	*# printing the word "Looping" until count's value is equals or greater than 3* count= 0 while count < 3: print ("Looping") count +=1

2.2.6 Modules: In Python, a module is a file containing reusable code, functions, classes, and variables that can be imported into other programs. Python comes with a standard library of built-in modules, but developers can also create custom modules or install third-party ones to extend functionality (Lutz, 2013).

- **Types of Modules**

Python modules can be categorized into three main types as shown in Table 14 below.

Table 14
Types of Modules

S/N	Types of Modules	Description	Syntax/ Example
1.	Built-in Modules	These modules are pre-installed with Python and provide essential functionality (R. Sharma, 2025)	Examples are: math (mathematical functions), random (random number generation), and sys (system-specific parameters and functions).
2.	User-Defined Modules	A user-defined module is a Python file created by a programmer containing functions, classes, or variables (N. Kumar, 2018)	Here is an example of a simple module named *my_module.py*, which will later be imported and used in another script *#my_module.py* def greet(name): return f"Hello, {name}!" *#using the my_module.py in another script:* import my_module print(my_module.greet("Alice")))
3.	Third-Party Modules	These are external modules developed by the Python community and can be installed using 'pip' in the code editor's terminal (Python's package manager) (Luna, 2019).	Examples include: NumPy (numerical computing), pandas (data manipulation), matplotlib (data visualization), and requests (handling HTTP requests)

- **Importing Modules in Python**

Modules can be imported in different ways depending on the requirements. Table 15 shows different ways of importing modules in Python.

Table 15
Ways of importing modules in Python

S/N	Ways of Importing Module	Description	Syntax/ Example
1.	Basic Import	This involves importing the Python built-in modules, such as the 'math' module	import math print (math.sqrt(100)) *# Output: 10.0*
2.	Import with an Alias	Occurs mostly when importing third-party libraries such as numpy	import numpy as np print(np.array([1, 2, 3]))
3.	Importing Specific Functions	When importing a specific function from a library using the *from ... import* ... syntax.	from math import sqrt print(sqrt(36)) *# Output: 6.0*

2.3. Python vs List Processing
Table 16.
Python vs LISP

S/N	Python	List Processing (LISP)

1	Python is renowned for its syntax, which is easy to understand, legible, and closely resembles natural language.
	
	It separates code blocks with indentation, which makes the code clear and easy to follow (Protasiewicz, 2024).
	
	Sample code:
	def add(a, b):
	  return a + b
		LISP is mostly made up of lists and has a more uniform and abstract syntax.
		
		LISP codes and data are written as s-expressions (symbolic expressions) with a prefix notation, which may not be clear and easy for beginners. (McJones, 2017)
		
		Sample code:
		(defun add (a b)
		  (+ a b))

2	Python supports object-oriented, functional, and procedural programming, making it suitable for various application use cases (Blackwell, 2024).	LISP supports procedural and object-oriented paradigms (for example, Common Lisp Object System -CLOS), making it efficient in meta-programming and symbolic computation. (McJones, 2017).
3	It is widely used in web development,	LISP is widely used in symbolic applications,

S/N	Python	List Processing (LISP)
	scripting, machine learning, data science, and automation (N. Sharma, 2023). Python is versatile for numerous applications because of its robust libraries such as NumPy, Pandas, and TensorFlow.	particularly in artificial intelligence research and natural language processing (Stephen M., 2022). It is also utilized in fields that need a great deal of customization, such as embedded systems, and in educational settings to teach programming fundamentals.
4	Generally faster for numerical computations. (GeeksforGeeks, 2024) Claimed Python is an interpreted language, which generally makes it slower than compiled languages. However, it is often fast enough for many applications, and performance can be improved with extensions like Cython or by using libraries written in C.	Can be slower for numerical computations but often faster for symbolic manipulation LISP implementations (like Common Lisp) can be compiled into machine code, leading to potentially higher performance than interpreted languages (Ihaka & Lang, 2008). LISP's macros also allow developers to optimi code in ways that are difficult in other languag
5	Its simplicity and libraries contribute to rapid development (N. Sharma, 2023). It's commonly used for prototyping and agile development due to its ease of use.	LISP enables rapid prototyping and incremental development. However, its steeper learning curve may slow down initial development for people not familiar with the language (Turner, 2012).
6	Python's simple syntax and readability make it a great choice for programming beginners (Khaled, 2023).	It is more challenging to learn, due to its distinctive syntax and abstract concepts like macros and functional programming (Stephen M., 2022).
7	Python supports metaprogramming to some extent through features like decorators, metaclasses, and introspection. However, it's more limited compared to LISP (Gupta, 2023).	LISP excels at metaprogramming, allowing programmers to manipulate code as data (homoiconicity) (Kumar, 2024). Its macro system is particularly powerful, enabling the creation of domain-specific languages (DSLs) and custom syntactic constructs.
8	Uses automatic memory management with reference counting and garbage collection (Tas, 2024).	Uses garbage collection to manage memory, but the exact mechanism may vary across different LISP dialects. (Shipley & Jodis,

S/N	Python	List Processing (LISP)
	Developers generally don't need to manually manage memory.	2003).
9	Supported by a wide range of IDEs and tools like PyCharm, VS Code, and Jupyter Notebooks (A. Gupta, 2020).	LISP often requires more specialized tools, with Emacs and SLIME being a common combination. However, modern IDE support for LISP is not as extensive as for Python. (Borretti & Nathan, 2024).
10	Large and active community, extensive libraries (NumPy, Pandas, TensorFlow, Scikit-learn) (N. Sharma, 2023).	Smaller community, but still has useful libraries (CL-NUMERIC, ACL2) (Stephen M., 2022).

2.4 Algorithm Comparison between Python and LISP

This section compares Python and LISP in implementing common algorithms: factorial calculation, Fibonacci sequence, summing a list, and reversing a list.

2.4.1. Factorial calculation: Both Python and Lisp use recursion for factorial. Python checks if *n == 0* and returns *1*; otherwise, it multiplies n by *factorial(n-1)*. Lisp does the same with *if* and returns *(* n (factorial (- n 1)))*. They are identical, but LISP uses verbose syntax and confusing parenthesis. This comparison is shown in Table 17. and Figure 2.

Table 17.
Algorithm comparison for factorial calculation.

Python	LISP

def factorial(n):
  if n == 0:
  return 1
  else:
  return n * factorial(n - 1)
print(factorial(5))
	(defun factorial (n)
	  (if (= n 0)
	  1
	  (* n (factorial (- n 1)))))
	;; Example usage
	(print (factorial 5)) ;; Output: 120

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```

```lisp
(defun factorial (n)
  (if (= n 0)
      1
      (* n (factorial (- n 1)))))

;; Example usage
(print (factorial 5))  ;; Output: 120
```

Fig. 2: *Python vs LISP implementation for factorial calculation*

2.4.2. Fibonacci Sequence: Python handles this with recursion, checking if *n <= 1* and returning *n*, otherwise recursively calling *Fibonacci (n-1) + Fibonacci (n-2)*. Lisp does the same but in a functional style, using *defun* and prefix notation, returning n for n <= 1 or recursively calling *Fibonacci (- n 1)* and *Fibonacci (- n 2)*. The comparison is displayed in Table 18. and Figure 3.

Table 18.
Algorithm comparison for Fibonacci sequence.

Python	LISP

def fibonacci(n):
  if n <= 1:
  return n
  else:
  return fibonacci(n - 1) + fibonacci(n - 2)
print(fibonacci(6))
# Output: 8 	
	(defun fibonacci (n)
	  (if (<= n 1)
	  n
	  (+ (fibonacci (- n 1)) (fibonacci (- n 2)))))
	;; Example usage
	(print (fibonacci 6))
	;; Output: 8

```python
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)

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

```lisp
(defun fibonacci (n)
  (if (<= n 1)
      n
      (+ (fibonacci (- n 1)) (fibonacci (- n 2)))))

;; Example usage
(print (fibonacci 6))  ;; Output: 8
```

Fig. 3: *Python vs LISP implementation for Fibonacci sequence*

2.4.3. A sum of a List: In Python, the sum of the list can be calculated in one line using the built-in *sum()* function. On the other hand, Lisp handles this operation through recursion, checking if the list is empty and then using the first element *(head)* and the rest of the list *(tail)* to sum the elements recursively. This comparison can be seen in Table 19. and Figure 4.

Table 19.
Algorithm comparison for the sum of a list.

Python	LISP

def sum_list(lst):
  return sum(lst)
print(sum_list([1, 2, 3, 4, 5]))
# *Output: 15* 	
	(defun sum_list (lst)
	  (if (null lst)
	  0
	  (+ (car lst) (sum_list (cdr lst)))))
	;; Example usage
	(print (sum_list '(1 2 3 4 5))) ;; Output: 15

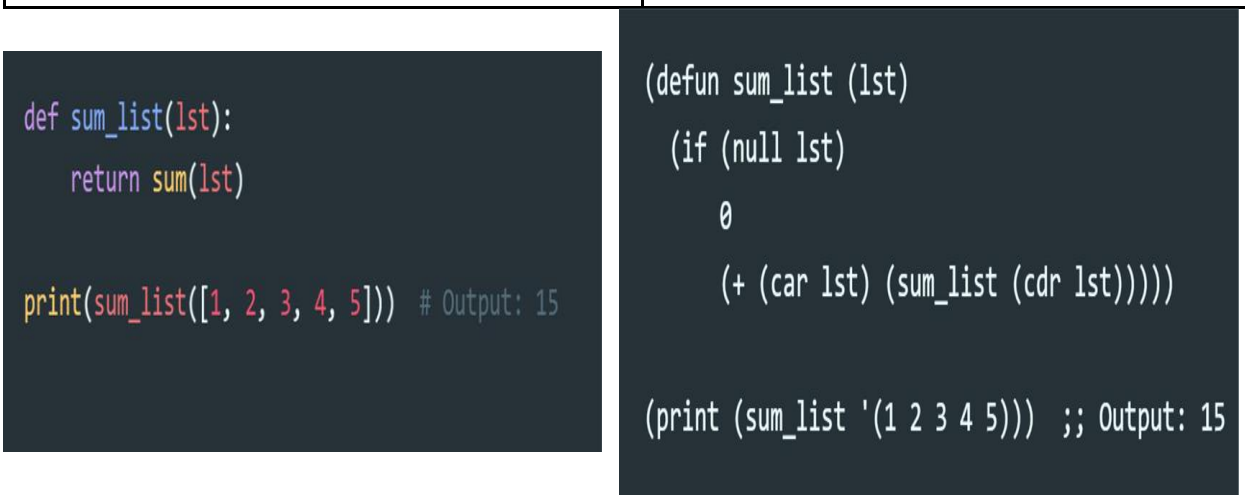

Fig. 4: *Python vs LISP implementation for the sum of a list*

2.4.4. Reverse a List: Python uses slicing *([::-1])* to reverse the list in one line, while LISP uses recursion, where it appends the first element (car) to the reversed remainder of the list (cdr). Python achieves this with a simple built-in feature, while Lisp manually constructs the reversed list recursively. This comparison is shown in Table 20. and Figure 5.

Table 20.
Algorithm comparison for reversing a list.

Python	LISP

def reverse_list(lst):
  return lst[::-1]
print(reverse_list([1, 2, 3, 4, 5]))
# Output: [5, 4, 3, 2, 1] 	
	(defun reverse_list (lst)
	  (if (null lst)
	  nil
	  (append (reverse_list (cdr lst)) (list (car lst)))))
	;; Example usage
	(print (reverse_list '(1 2 3 4 5)))
	;; Output: (5 4 3 2 1)

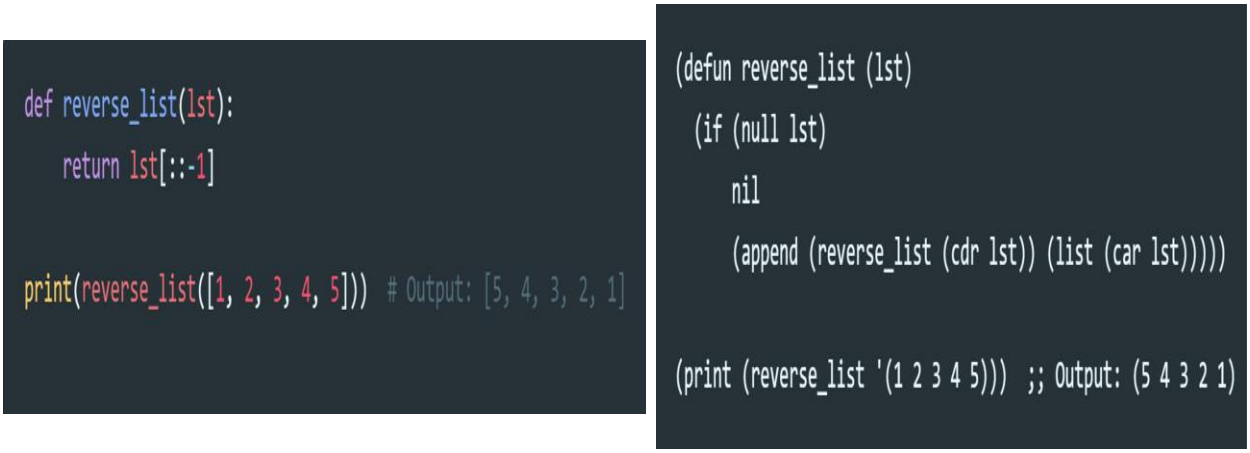

Fig. 5: *Python vs LISP implementation for reversing a list.*

2.5 The Adoption and Integration of Python with AI and ML.
Python became the most preferred programming language in Artificial Intelligence and Machine Learning due to its simplicity, versatility, and extensive ecosystem of libraries and frameworks.

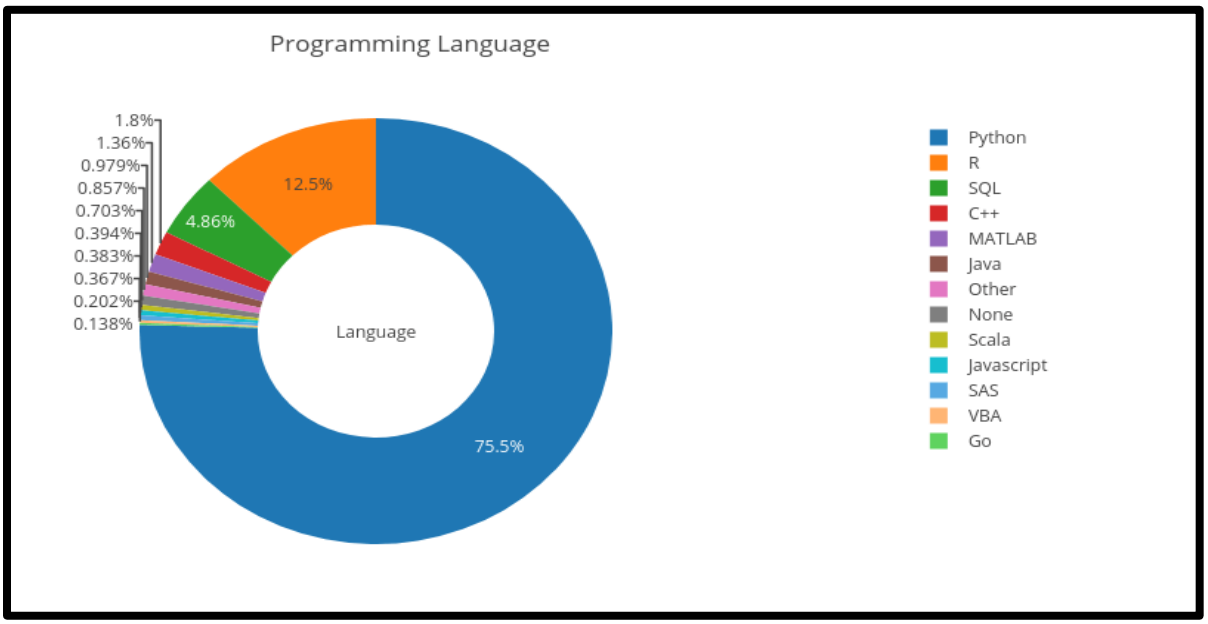

Fig. 6: *Survey on the most preferred programming language by 14,000 developers (Quintagroup, 2019).*

Its adoption in AI and ML has revolutionized the field, enabling rapid development, prototyping, and deployment of intelligent systems (McRorey, 2025). This section is focused on the factors that drove Python's adoption in AI and ML and how its integration process.

2.5.1 Factors that Drove Python's Adoption in AI system

The adoption of Python into AI/ML was driven by factors like syntax, which is intuitive and closely related to natural language; it made Python more accessible to both beginners and experienced developers. Its code's simplicity and readability brought about the reduced time required to draft and debug code, which has helped a lot in the development of AI and ML projects (Prabu et al, 2024). Also, the availability of extensive library frameworks, such as TensorFlow, PyTorch, Scikit-learn, Keras, and Pandas, among others, in Python programming has significantly contributed to its preference for AI and ML over other programming languages. The aforementioned factors have been useful in streamlining complex tasks in terms of data preprocessing, model training, and evaluation; they have also enabled AI and ML developers to solve problems quickly by leveraging the existing information that is available to them (McKinney, 2017).

Notably, AI and ML models built using Python can operate across multiple operating systems with minimal adjustments, making it highly adaptable for deployment in diverse environments. Python also integrates effortlessly with other programming languages and technologies, such as C/C++ for enhancing performance and JavaScript for web-based applications. This interoperability significantly boosts its effectiveness in AI and ML workflows, ensuring flexibility and efficiency (Ghimire, 2020).

Beyond its technical strengths, Python benefits from a large and active community that drives its ongoing development. This community contributes not only to extensive documentation but also to tutorials, forums, and resources that help developers learn and resolve issues. Such support

fosters innovation and collaboration, further solidifying Python's position as a leading tool in AI and ML (Blackwell, 2024).

2.5.2 From Data to Deployment

Python offers an integrated ecosystem that seamlessly connects every stage of development. Libraries like Pandas and NumPy simplify data preprocessing, enabling efficient data manipulation, cleaning, and analysis-essential steps for preparing datasets. These tools work hand-in-hand with frameworks like TensorFlow and PyTorch, which provide high-level APIs for building and training models, from traditional ML algorithms to advanced deep learning architectures. This smooth transition from data preparation to model development highlights Python's ability to unify diverse tasks into a single, cohesive workflow (Prabu et al, 2024).

Beyond Python's integration in the development of AI and ML, its integration extends to visualization and deployment with the use of Libraries like Matplotlib and Seaborn which allows developers to create insightful visualizations, making it easier to interpret results and communicate findings. For deployment, tools like Flask and Django enable models to be integrated into web services, while libraries like Dask and Ray ensure scalability for handling large datasets and complex computations. This end-to-end integration from data preprocessing to deployment, makes Python an indispensable tool for AI and ML workflows, ensuring efficiency and coherence across all stages (Ghimire, 2020).

3.O THE ADVANCED COMPONENTS OF PYTHON AND THEIR USES IN AI AND ML.

Python has several game-changing features that make it an essential tool for AI and ML. These features include:

- **Context Managers:** These provide a structured and reliable way to manage resources and ensures that setup and teardown operations are executed automatically even when exceptions come up. These are implemented using the with statement which calls the **__enter__** method when entering the block and the __exit__ method when exiting. In AI/ML, it is used to manage file resources like datasets, model weights and training logs (Khare, 2024).

```
class SimpleContextManager:
    def __enter__(self):
        print("Entering context")
        return self  # Important: Return self or the resource

    def __exit__(self, exc_type, exc_val, exc_tb):
        print("Exiting context")

with SimpleContextManager() as manager:
    print("Inside the 'with' block")
    # You can access the context manager instance here if needed (using 'manager'
print("Outside the 'with' block")
```

Fig. 7: *Code snippet to illustrate context managers in Pyth*on

- **Metaclasses:** These are used for controlling the creation and behaviour of classes and are often referred to as "Classes of classes". They are used to define how class objects are

instantiated which allows dynamic modification of class attributes and methods. Some of these classes can be created by inheriting from default metaclass, type, and overriding methods like __new__ and __init__ to customize the class creation. In AI/ML, they are used for automatic registration of models and components which allows subclasses created to be automatically registered in a central repository (Rafalski, 2024).

```
class Meta(type):
    def __new__(cls, name, bases, dct):
        dct['attr'] = 'Added by Meta'
        return super().__new__(cls, name, bases, dct)

class MyClass(metaclass=Meta):
    pass

print(MyClass.attr)
```

Fig. 8: *Code snippet to illustrate metaclasses in python*

- **Abstract Base Classes (ABCs):** ABCs are provided by the abc module in Python and they serve as blueprints for creating interfaces and ensuring that concrete subclasses use specific sets of methods. They are defined by inheriting from abc.ABC and using the @abstractmethod decorator to declare methods that must be implemented by any non-abstract subclass. In defining interfaces between different components of a system like models, data loaders, etc. ABCs are useful in such AI/ML applications (Jain, 2024).

```
from abc import ABC, abstractmethod

class MyABC(ABC):
    @abstractmethod
    def my_method(self):
        pass

class ConcreteClass(MyABC):
    def my_method(self):
        print("Implementation of my_method")

b = ConcreteClass()
b.my_method()
```

Fig. 9: *Code snippet to illustrate abstract base classes in python*

- **Multiple Inheritance and MRO (Method Resolution Order):** Python has a feature which allows a class to inherit from more than one parent class - this is called Multiple Inheritance. Thica can be useful for creating specialized model classes that inherit functionalities from a base class and also uses specific features from mixin classes. When a class inherits from multiple parents, Method Resolution Order (MRO) is a mechanism that determines the order in which methods and attributes are searched for in the class hierarchy. MRO follows a depth-first, left-to-right approach. Developers can inspect the MRO of a class by using __mro__ attribute or the mro() method (Rohith, 2023).

```
class A:
    def method(self):
        print("Method from A")

class B:
    def method(self):
        print("Method from B")

class C(A, B):
    pass

c = C()
c.method()  # Calls A's method
print(C.__mro__)  # Shows the method resolution order
```

Fig. 10: *Code snippet to illustrateMultiple Inheritance and MRO in python*

- **Coroutines and Async Programming:** Python has a library called asyncio which is responsible for handling coroutines and asynchronous programming and it uses the async/await syntax. This functionality provides tls for mproviing the efficiency of I/O bound operations in AI/ML. Often times, when creating AI/ML pipelines, data loading and preprocessing, these operations involve significant I/O. Asynchronous programming allows the program to perform other tasks while waiting for I/O operations to complete and this is great for improving performance (Gangopadhyay, 2025).

```
import asyncio

async def my_coroutine():
    print("Coroutine started")
    await asyncio.sleep(1)
    print("Coroutine finished")

async def main():
    await my_coroutine()

asyncio.run(main())
```

Fig. 11: *Code snippet to illustrate Coroutines and Async Programming in python*

- **Dynamic Code Execution (exec and eval)**: Python has functions called exec() and eval() which gives the ability to execute Python code dynamically from strings. In AI/ML, this is useful for rapid prototyping because it allows researchers to quickly test new ideas oor run experiments with dynamical code snippets. Table 21 below shows a comparison of exec() and eval() (Jenifar, 2023).

Table 21.
Dynamic Code Execution (exec and eval)

Feature	exec()	eval()
Purpose	Execute a block of statements	Evaluate a single expressiion
Return value	Returns None	Returns the result of the expression
Code type	For blocks of code including statements	For expressions
In AI/ML	Dynamic model construction based on configurations	Dynamical evaluation of expressions (e.g loss functions)

```
x = 10
y = 20
# Execute a statement using exec
exec("z = x + y")
print(z)   # Output: 30

# Evaluate an expression using eval
result = eval("x * y")
print(result)   # Output: 200
```

Fig. 12: *Code snippet to illustrate exec() and eval() in python*

- **Concurrency with Threading and Multiprocessing:** These modules (threading and multiprocessing) provide mechanisms for achieving concurrency and parallelism in AI/ML. Threading is when concurrent execution of tasks are allowed within a single process while multiprocessing is when true parallelism is enabled by using multiple CPU cores and bypassing the Global Interpreter Lock (GIL). These functions make it easy to handle CPU-intensive tasks like model training and many libraries like scikit-learn provide support for multiprocessing (Premanand, 2024).

```
import threading
import multiprocessing

def thread_function():
    print("Thread running")

def process_function():
    print("Process running")

# Threading
thread = threading.Thread(target=thread_function)
thread.start()
thread.join()

# Multiprocessing
process = multiprocessing.Process(target=process_function)
process.start()
process.join()
```

Fig. 13: *Code snippet to illustrate threading and multiprocessing in python*

- **Decorators**: These are special functions that can modify the characteristics of other functions (Luciano, 2022). Figure 14. shows an example of a decorator function.

```
def decorator(func):
    def wrapper():
        print("Before function")
        func()
        print("After function")
    return wrapper

@decorator
def say_hello():
    print("Hello!")

say_hello()
```

Fig. 14. *An example of a decorator function*

- **Generators**: These are tools that simplify code and enhance performance, particularly in handling complex data problems (Sunil, 2019). Generators yield items one at a time for memory conservation. As seen in Figure 15, we use the 'yield' keyword to achieve this task.

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

for num in countdown(5):
    print(num)
```

Fig. 15: *Generator using 'yield'*

4.0 DISCUSSION

4.1 The Trajectory of AI and ML with the Python Programming Language

Python came into dominance in the fields of Artificial Intelligence and Machine Learning of its user-friendliness, itchfree syntax, and the extensivity of it's libraries and frameworks, which made it the most preferred programming language for both academic researchers and professional AI developers (Sharma, 2023). The in-depth study conducted by (Pranav & Sarma, 2023) stated that as AI and ML continue to advance, it would leverage Python's features to achieve its expansion; it would shape the emerging trends and play a crucial role in defining the future trajectory of these domains. Figure 16 provides a historical perspective on the evolution of ML and AI, outlining its progression through three generations: "The Backend", "The Human Side", and "Pattern Recognition" while also offering insights into anticipated future developments.

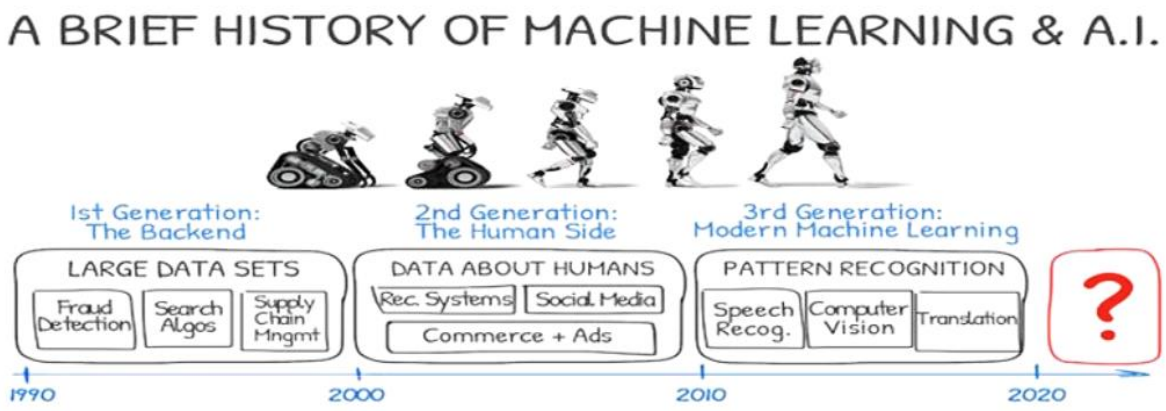

Fig. 16: *Brief History of Machine Learning and A.I (Pranav & Sarma, 2023)*

Figures 17. and 18. show a brief timeline of the most important events in the history of machine learning and artificial intelligence respectively.

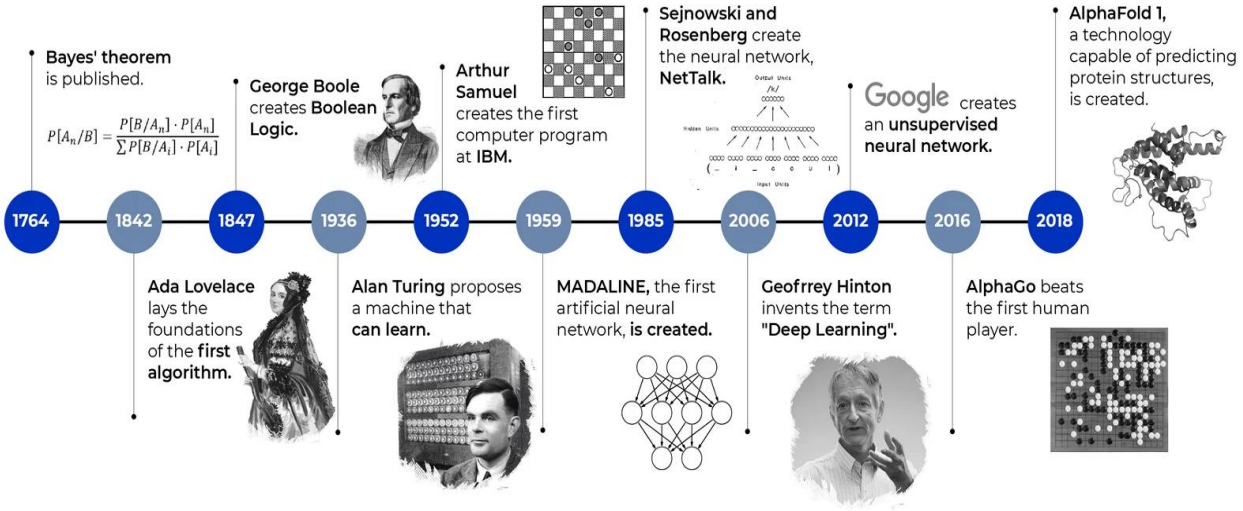

Fig. 17: *Key Events in the History of Machine Learning (Pranav & Sarma, 2023*)

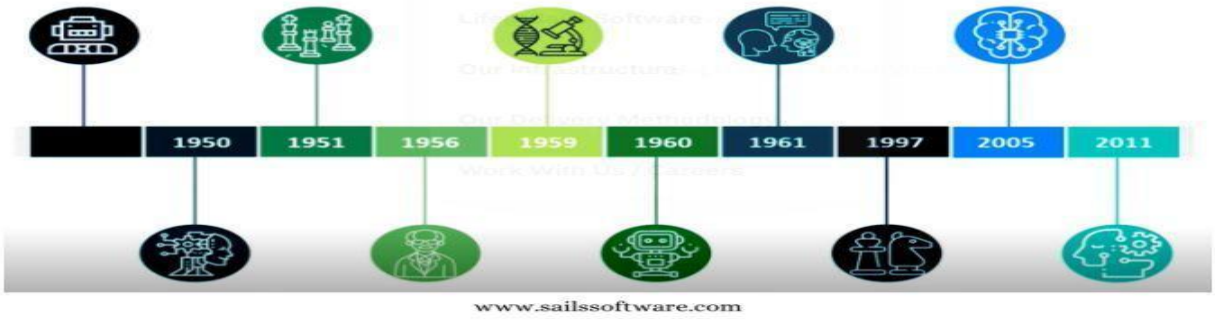

Fig. 18: *Timeline of Artificial Intelligence Timeline (Bhargavi, 2022)*

4.1.1 Growth in Deep Learning Applications: One of the most notable developments in AI/ML is the ongoing growth of deep learning. Python, with its libraries such as TensorFlow, PyTorch, and Keras, has become closely associated with deep learning development. As noted by (Sharma et al. 2024), these libraries offer tools for developing and training neural networks and support advanced architectures like transformers, GANs (Generative Adversarial Networks), and reinforcement learning models. Their seamless integration with Python has increased both research and application in fields like natural language processing (NLP), computer vision, and autonomous systems. Figure 19. highlights some Python-based deep learning applications.

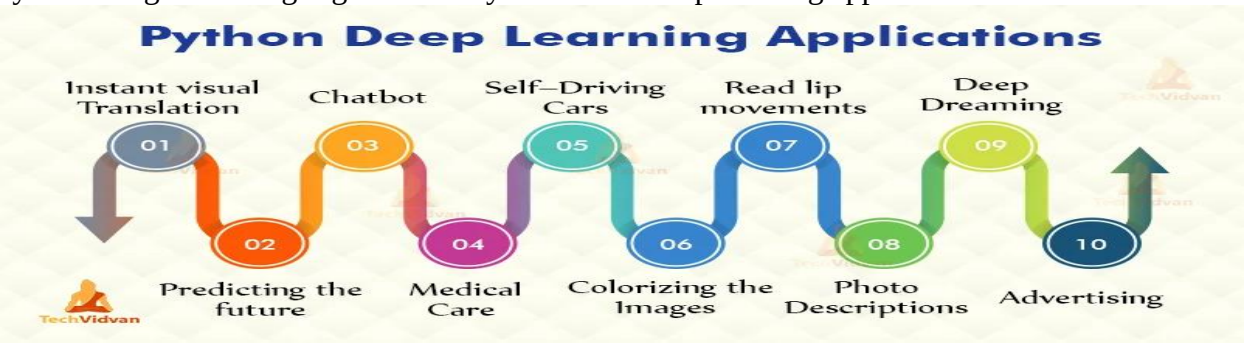

Fig. 19: *Python Deep Learning Applications (Vidvan, 2020)*

***4.1.2 Evolution of Automated* Machine Learning (AutoML) and Model Optimization:** AutoML is an emerging trend aimed at automating the end-to-end process of applying ML to real-world problems (Salehin et al., 2024). Python is central to this trend, with tools like AutoSKlearn and TPOT offering frameworks that simplify model selection, hyperparameter tuning, and feature engineering. As AutoML evolves with Auto-Keras, Python's role is expanding to include more sophisticated model optimization techniques, such as neural architecture search (NAS), which automatically discovers optimal neural network architectures (Jin et. al., 2019).

4.1.3 Integration with Big Data and Cloud Computing: The convergence of AI/ML with big data and cloud computing is another emerging trend where Python is at the forefront. Python's integration with big data frameworks like Apache Spark (via PySpark) and its compatibility with cloud services such as AWS, Google Cloud, and Azure, allow for scalable AI/ML solutions (Rahman & Rana, 2021). This integration is crucial for handling the massive datasets that modern AI/ML models require. Moreover, the rise of cloud-based AI services, many of which provide Python APIs, is making it easier to deploy and manage AI/ML models at scale.

4.1.4 Democratization of AI and ML: Python is playing a pivotal rolein making artificial intelligence (AI) and machine learning (ML) more accessible to a wider audience. With user-friendly libraries such as Scikit-learn and the growing availability of educational resources and online platforms, the entry barriers for understanding and applying ML have significantly decreased. This accessibility is encouraging innovation across various domains such as healthcare (for personalized treatment), education (for adaptive learning platforms), and marketing (for targeted customer segmentation) by enabling individuals without deep technical backgrounds to develop AI-powered solutions. Recent studies by Turley (2024) highlights the expanding role of low-code and no-code platforms, many of which are often powered by NJ Python in the backend.

4.1.5 Advancements in Explainability and Interpretability: The high complexity of AI and machine learning models makes the demand for explainability and interpretability more important for the wider adoption of AI applications in regulated sectors such as healthcare and finance. Python is leading advancements in this explainability with libraries like SHapley Additive exPlanations (SHAP) and Local Interpretable Model-agnostic Explanations (LIME) (Linardatos et al., 2020). These libraries enable researchers and professionals to comprehend model predictions better and also, enhance the transparency and trustworthiness of AI systems.

4.1.6 Reinforcement Learning and Autonomous Systems: Reinforcement learning (RL) is a type of machine learning process that focuses on decision-making by autonomous agents. An autonomous agent is an independent system that can perceive its surroundings, make decisions based on that perception, and take action to achieve its goals without constant human intervention (Murel & Kavlakoglu, 2024). RL is a significant focus in machine learning research and is increasingly applied in fields such as robotics, autonomous systems, and gaming. Python is the primary language for conducting RL experiments and implementations, largely due to its compatibility with RL libraries such as OpenAI Gym, Stable Baselines, and RLlib, which are driving progress in the field (Sutton et al., 2018). Figure 38. features CARLA, an open-source driving simulator with a Python API used in autonomous driving research.

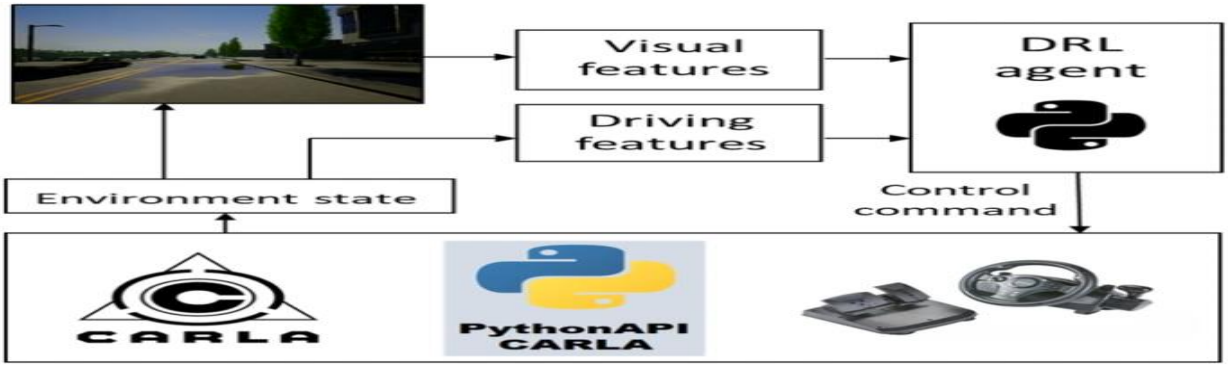

Fig. 20: *Python API CARLA (Pérez-Gil et al., 2022)*

4.1.7 AI Ethics and Responsible AI Development: The ethical considerations surrounding AI/ML technologies are receiving more attention, and Python plays a key role in advancing responsible AI frameworks. Libraries such as Fairlearn and AI Fairness 360 (AIF360) offer resources to evaluate and reduce bias in AI models (Bellamy et al., 2019). Tools like AIF360, FairLearn, and SHAP (SHapley Additive exPlanations) are becoming more commonly incorporated into AI/ML workflows, highlighting the growing movement toward ethical AI practices. These tools are increasingly embedded within enterprise and cloud-based platforms such as IBM Watson OpenScale and Microsoft Azure Machine Learning (which integrates FairLearn and SHAP for model fairness and explainability).

4.2 The Imperative Benefits of Using Python for AI and ML.

The imperative benefits of Python programming language in Artificial Intelligence and Machine Learning are numerous and are discussed below:

4.2.1 Stability, Flexibility, and Simplicity: Python has proven to be the ideal language for AI and ML applications due to its stability, flexibility, and simplicity because it allows developers to write reliable, readable code and prototype quickly (Worsley, 2024). The focus of Python programming language on problem-solving and its open-source license, which makes its use free and distributivity, made it an important tool in AI and ML (Mihajlovic et. al., 2020).

4.2.2 Community Dependability: Python has a very dynamic community that provides strong support that is useful in the development of AI and ML. Its open-source language feature makes numerous resources available to AI and ML developers across all levels of proficiencies. Diverse Python problem-solving forums that are available to help AI and ML developers have consistently proven to be valuable in rendering help and providing solutions to any type of Python problem (Ryabtsev, 2024).

4.2.3 Extendable Libraries: Python has a very vast ecosystem of libraries such as Scikit-learn, TensorFlow, Keras, Numpy, Pytorch, and Cython which are powerful tools for building and deploying machine learning models, making Python indispensable for cutting-edge technology development. Scikit-Learn is useful in processing data, and writing classical programs and also serves as a vital tool in feature engineering, TensorFlow is a crucial tool used in large AI applications, model deployment, and deep learning, and Keras helps achieve rapid prototyping in a high-level neural network, NumPy is best applied in matrix operation and numerical computations into PyTorch is a vital tool in the development and achievement flexible AI/ML models, and Cython is used for the optimization of python to ensure improved performance (Singh, 2025).

4.2.4 Increased Multifunctionality: Python provides the choice of using object-oriented programming or scripting which enables AI/ML developers to modify processes without the need to compile the source code again. It provides a flexible work environment for AI Engineers and enables them to use their preferred programming style, which results in quick problem resolution. Python helps to eliminate extreme coding burdens from AI and ML developers through its multifunctionality (Dergano, 2023).

4.2.5 Popularity and Dominance: the trajectory of Python in AI and ML has shown that Python programming language has become widely recognized and adopted worldwide, which makes available more population of AI/ML developers and gives room for the ease of recruiting new team members and for the expansion of AI and ML inventions through the availability of sufficient workforce (Ryabtsev, 2024).

4.2.6 Strong Visualization and Data Analysis Tools: Python offers different data visualization and analysis libraries, which help researchers interpret complex datasets efficiently. Tools like Matplotlib, Seaborn, and Pyecharts simplify the development of visualizations, improving how data is understood and conveyed to stakeholders (Cao et al., 2021). Additionally, Python supports essential data analysis tasks such as cleaning, wrangling, and modeling, which aid in exploratory data analysis (EDA) and discovering meaningful insights (Singh et al., 2022). These characteristics are an important step in developing powerful and reliable AI/ML models.

4.2.7 Support for Parallel and Distributed Computing: Python also has great ability in the parallel and distributed computing space which is important for handling large-scale computations. This capability is crucial in developing deep learning models and large-scale AI/ML applications. Python has tools such as MPI(mpi4py) - Message Passing Interface - a standardized, portable message-passing system designed to work on a wide variety of parallel computers, PETSc (petsc4py) - Portable Extensible Toolkit for Scientific Computation - a collection of algorithms for the solution of problems in the area of engineering and scientific applications especially relating to partial differential equations for high-performance parallel computers (Dalcin et al., 2011).

4.2.8 Integrability with other languages: Oftentimes, while building AI/ML solutions, we sometimes need to leverage the speed of low-level languages while maintaining simplicity and this is why Python is a great language for this task. Python was designed to allow easy integration and portability with other languages such as C, C++, Fortran, etc. This functionality allows developers to optimize components that are performance-critical in their applications. We have libraries dedicated to this, an example of such is Cython which compiles Python code to C for improved execution speed and interfaces that allow embedding Python within C/C++ programs (Behnel et al., 2011). This is a flexibility that combines Python's ease of usage with the efficiency of lower-level languages like C/C++.

4.2.9 Impact on the advent of Large Language Models (LLMs): Python has been fundamental to the development and training of complex neural networks, thanks to powerful frameworks like TensorFlow and PyTorch. These tools provide great support for building large Transformer models like BERT (Bidirectional Encoder Representations from Transformers). Many leading LLMs, such as BERT and GPT, were created using TensorFlow or PyTorch, these are possible due to Python's simple syntax, which makes it easier to experiment with model architectures and training processes (Marija & Dražen, 2022). Furthermore, Python has greatly contributed to making LLMs more accessible, especially through projects like the Hugging Face Transformers library, which brought a a lot of Transformer models together under a single, easy-to-use API (Wolf et al., 2020).

4.2.10 Automation: Python is commonly adopted as a tool to automate repetitive tasks due to its simplicity and clarity, this means that even complex procedures can be written with simplicity and iterated to solve complex problems with little effort. Python codes can be written into scripts used to glue together steps in a workflow, for example, fetching data, processing it with an AI model, and triggering some actions even without human intervention. With this advantage, humans can focus on higher-level work while repetitive tasks are done automatically and this leads to higher productivity and effectiveness (Karabulut & Akyuz, 2023).

4.2.11 Ease of Learning and Readability: Python's syntax is structured to be clear and easy to understand, which makes it more accessible for beginners and promotes seamless collaboration across diverse teams (Lutz, 2013). Its readability simplifies the learning process and is especially beneficial in AI and ML projects or research, where well-organized and maintainable code is crucial for ensuring reproducibility and scalability.

4.2.12 Support for Emerging Technologies: Python has established itself as one of the go-to language options for emerging fields like quantum computing, edge AI, and federated learning

because of its versatility and rich ecosystem of libraries. Tools such as Qiskit (Abraham et al., 2019) and Cirq (Google Quantum AI Team, 2020) support quantum computing research, while TensorFlow Federated (Ingerman & Ostrowski, 2019) facilitates federated learning experiments. Its flexibility and seamless integration with advanced frameworks make Python an ideal programming language choice for researchers exploring cutting-edge applications in AI and ML.

4.2.13 Rapid Prototyping and Experimentation: The dynamic typing and interpreted nature of Python enable researchers to quickly develop prototypes and test ideas, which plays a crucial role in the iterative AI and ML development process (Pérez et al., 2011). This feature facilitates rapid experimentation and hypothesis validation, significantly reducing the time required to gain insights into research projects. Additionally, tools like Jupyter Notebooks enhance this workflow by offering an interactive platform for iterative development and real-time feedback (Kluyver et al., 2016).

4.2.14 Ethical AI Development and Collaboration: Python's open-source ecosystem and community-driven approach enhance transparency and collaboration, which are essential for tackling ethical challenges in artificial intelligence and machine learning, including bias, fairness, and accountability (Weerts et al., 2023). Frameworks such as Fairlearn (Bird et al., 2020) and AI Fairness 360 (Bellamy et al., 2018), built on Python, empower researchers to detect and mitigate biases in AI models. This supports the increasing focus on responsible AI research and the establishment of ethical guidelines in the field.

4.2.15 Accessibility for Non-Programmers: Python's clear and intuitive syntax makes it a preferred option for researchers and professionals with limited programming experience. Its user-friendly structure and comprehensive documentation simplify the learning process, allowing individuals from various fields to engage in AI and ML research (Lutz, 2013). This accessibility promotes interdisciplinary collaboration and expands opportunities for innovation in the field**.**

5.0 CONCLUSION

The fields of AI and ML have reached impressive levels of dominance and preference across various industries, primarily due to their reliance on the advanced features of the Python programming language. Key aspects such as context managers, metaclasses, abstract base classes, multiple inheritance, method resolution order (MRO), coroutines, asynchronous programming, dynamic code execution (using exec and eval), as well as concurrency through threading and multiprocessing, and the use of generators with decorators have all contributed to this success. Over time, AI and ML have experienced a dynamic trajectory, marked by growth in deep learning applications, optimization of automated machine learning and its models, integration with big data and cloud computing, and the democratization of technology, with absolute kudos to Python.

In conclusion, the invaluable features and advantages of the Python programming language—including stability, flexibility, simplicity, community support, an extensive library ecosystem, enhanced multifunctionality and interoperability, widespread popularity, robust visualization, and data analysis tools, support for parallel and distributed computing, capability for integration with other languages, and its significant impact on the emergence of large language models (LLMs)—make it an essential programming language in the realm of AI and ML. Additionally, its support for automation, ease of learning and readability, advocacy for ethical AI development and

collaboration, and accessibility for non-programmers further solidify its critical role in this domain.

6.0 RECOMMENDATIONS

Python's dynamic prototyping which can lead to runtime errors that might not exist in more statically typed languages, and potentially undermine security and software reliability in AI applications should be addressed. Python's simplicity which might limit the depth of optimization and performance tuning compared to languages traditionally that could be used in AI and ML should be addressed.

Future research should assess the challenges of Python in AI and ML, such as performance limitations for computationally intensive tasks and dependency management in large projects. Improved support should be provided for parallel computing and enhanced integration with emerging technologies like quantum computing, which will further strengthen its role in AI and ML.

Future studies should focus on developing advanced tools for automated testing, building secure machine learning pipelines, and supporting privacy-focused methods like federated learning with tools such as PySyft.

7.0 REFERENCE

Abraham, H., et al. (2019). Qiskit: An Open-source Framework for Quantum Computing. Qiskit Documentation.

Adetiba, E., John T.M., Akinrinmade, A.A., Moninuola, F.S. , Akintade O.O., and Badejo J.A. (2021). Evolution of artificial intelligence languages – a systematic literature review. Journal of Computer Science. 17(11), 1157-1171. http://dx.doi.org/10.3844/jcssp.2021.1157.1171

Alex Ryabtsev. (2024). 8 Reasons Why Python is Good for AI and ML. Retrieved March 7, 2025, from https://djangostars.com/blog/why-python-is-good-for-artificial-intelligence-and-machine-learning/

Ali, M. (2015). Python data types explained: A beginner's guide. DataCamp. Retrieved March 24, 2025, from https://www.datacamp.com/blog/python-data-types

Bellamy, R. K. E., Dey, K., Hind, M., Hoffman, S. C., Houde, S., Kannan, K., & Zhang, Y. (2019). AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM Journal of Research and Development*, *63*(4/5),.1-2 https://doi.org/10.1147/jrd.2019.2942287

Bhargavi. (2022, October 25). Artificial intelligence history, stages, types, and domains. Retrieved from https://sailssoftware.com/artificial-intelligence/

Blackwell, A. (2024). Moral Codes: Designing Alternatives to AI. https://doi.org/10.7551/mitpress/14872.003.0012

Blackwell, Alan. (2024). Explanation and transparency: Beyond no-code/low-code. In Moral Codes (pp. 99–116). The MIT Press. Retrieved from http://dx.doi.org/10.7551/mitpress/14872.003.0010

Borretti, F., & Nathan, P. (2024). Using Emacs as an IDE. Retrieved from https://github.com/LispCookbook/cl-cookbook/blob/master/emacs-ide.md website: https://lispcookbook.github.io/cl-cookbook/emacs-ide.html

Brundage, M., Avin, S., Wang, J., Belfield, H., Krueger, G., Hadfield, G., … Anderljung, M. (2020, April 15). Toward trustworthy AI development: Mechanisms for supporting verifiable claims. Retrieved April 23, 2025, from arXiv.org website: https://arxiv.org/abs/2004.07213

Cao, S., Zeng, Y., Yang, S., & Cao, S. (2021). Research on Python data visualization technology. *Journal of Physics: Conference Series, 1757*(1), 012122. https://doi.org/10.1088/1742-6596/1757/1/012122

Chatterjee, S. (2024, February 28). Master float in Python: The ultimate guidebook to precision. Emeritus. Retrieved March 24, 2025, from https://emeritus.org/in/learn/float-in-python/

Chen, Y., & Huang, L. (2025). Conditional statements. Retrieved from Springer, pp 133–152 website: https://link.springer.com/chapter/10.1007/978-981-97-8788-3_4

Dalcin, L. D., Paz, R. R., Kler, P. A., & Cosimo, A. (2011). Parallel distributed computing using Python. *Advances in Water Resources*, *34*(9), 1124–1139. http://dx.doi.org/10.1016/j.advwatres.2011.04.013

Dergano F. (2023). Python's Role in Artificial Intelligence and Machine Learning. Retrieved February 27, 2025, from https://raccomandino.medium.com/pythons-role-in-artificial-intelligence-and-machine-learning-b6b97843a307

Gaddis, T. (2024). Step-by-step solution. *Vaia*. Retrieved from https://www.vaia.com/en-us/textbooks/computer-science/starting-out-with-c-from-control-structures-through-

objects-8-edition/chapter-5/problem-19-the-statement-or-block-that-is-repeated-is-known- (pp.)

Gangopadhyay, S. (2025). *The asynchronous IO revolution: How Python is changing the game*. GUVI. https://www.guvi.in/blog/the-asynchronous-io-revolution-how-python-is-changing-the-game/ (Retrieved April 12, 2025)

GeeksforGeeks. (2024). Why Python is Called Interpreted Language. GeeksforGeeks. https://www.geeksforgeeks.org/why-python-is-called-interpreted-language/Geeksforgeeks (2025). Python Operators. Retrieved from March 25, 2025, from https://www.geeksforgeeks.org/python-operators

Ghimire, D. (2020). Comparative study on Python web frameworks: Flask and Django. https://www.theseus.fi/bitstream/handle/10024/339796/Ghimire_Devndra.pdf?sequence=2

Google Quantum AI Team. (2020). Cirq: A Python Framework for Creating, Editing, and Invoking Noisy Intermediate Scale Quantum (NISQ) Circuits. Cirq Documentation.

Gupta, A. (2023). Machine Learning with Python; Retrieved from https://medium.com/@aaditgupta06/machine-learning-with-python-aedc5353a72a

Gupta, A. (2020, September 30). 20 most popular Python IDEs in 2024: Code like a pro. Simplilearn. Retrieved from https://www.simplilearn.com/tutorials/python-tutorial/python-ide

Gupta, N. (2023, March 27). Meta-programming in Python: Unleashing the power of decorators, metaclasses, and introspection. DataScience with Python—NishKoder. Retrieved from https://medium.com/nishkoder/meta-programming-in-python-unleashing-the-power-of-decorators-metaclasses-and-introspection-e1274c753dc1

Haenlein, M., & Kaplan, A. (2019). A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. *California Management Review*, *61*(4), 5–14. https://doi.org/10.1177/0008125619864925

Ihaka, R., & Lang, D. T. (2008, January 1). Back to the Future: Lisp as a base for a statistical computing system. (pp. 5-6) https://www.researchgate.net/publication/227019917_Back_to_the_Future_Lisp_as_a_Base_for_a_Statistical_Computing_System

Ingerman, A., & Ostrowski, K. (2019). TensorFlow Federated: Machine Learning on Decentralized Data. TensorFlow Blog.

Jain, A. (2024). Embracing abstraction: A dive into abstract classes in Python. Medium. https://medium.com/@abhishekjainindore24/embracing-abstraction-a-dive-into-abstract-classes-in-python-0faf6d83948d (Retrieved April 12, 2025)

Jenifar. (2023). *Python exec vs eval*. Medium. https://medium.com/@brusooo27/python-exec-vs-eval-ea949931ee8f (Retrieved April 12, 2025)

Jin, H., Song, Q., & Hu, X. (2019). Auto-Keras: An efficient neural architecture search system. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York, NY, USA: ACM. Retrieved from http://dx.doi.org/10.1145/3292500.3330648 (pp. 1947)

Karabulut, N., & Akyüz, Y. (2023). *A Comparison of RPA Tool and Python Programming Language for a BOM Digitization Project in Automobile Industry*. *Yönetim Bilişim Sistemleri Dergisi*, *9*(2), 15-26. http://dergipark.gov.tr/ybs

Khaled, M. (2023, December 7). Python programming language: A comprehensive overview. Retrieved from ProfileTree Web Design and Digital Marketing website: https://profiletree.com/python-programming-language-comprehensive-overview/

Khare, Y. (2024). *Use cases of Python context manager*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2024/06/python-context-manager/ (Retrieved April 12, 2025)

Kuchling, A. M. & Zadka, M. (October 2000). "What's New in Python 2.0". Python Software Foundation. Retrieved November 20, 2024, from https://docs.python.org/3/whatsnew/2.0.html

Kuhlman, D. (September 2011). A Python Book: Beginning Python, Advanced Python, and Python Exercises. Section 1.1. Retrieved November 20, 2024, from https://www.academia.edu/resource/work/34001890

Kluyver Thomas, Ragan-Kelley Benjamin, Perez Fernando, Granger Brian, Bussonnier Matthias, Frederic Jonathan, … Jupyter Development Team. (2016). Jupyter Notebooks-; a publishing format for reproducible computational workflows. In Positioning and Power in Academic Publishing: Players, Agents, and Agendas (pp.87-90). IOS Press. https://doi.org/10.3233/978-1-61499-649-1-87

Kumar, R. (2024). Power of LISP for language-oriented programming. Retrieved from https://industrywired.com/power-of-lisp-for-language-oriented-programming/

Kumar, N. (2018). Python modules tutorial: Importing, writing, and using them. DataCamp. Retrieved from https://www.datacamp.com/tutorial/modules-in-python

Lily Hulatt & Grabriel Frietas, (2024). Programming Paradigms. Retrived from https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/programming-paradigms/

Linardatos, P., Papastefanopoulos, V., & Kotsiantis, S. (2020). Explainable AI: A review of machine learning interpretability methods. Entropy, 23(1), 18. https://doi.org/10.3390/e23010018

List Processor for Businesses - What It is and How to Use It. (n.d.). Retrieved from Lenovo UK website: https://www.lenovo.com/gb/en/glossary/lisp/

Luna, J. C. (2019). Pip Python tutorial for package management. DataCamp. Retrieved from https://www.datacamp.com/tutorial/pip-python-package-manager

Lutz, M. (2013). *Learning Python: Powerful object-oriented programming*. "O'Reilly Media, Inc." (pp. 3-4)

McJones, P. R. (2017, October 1). The LISP 2 Project. Retrieved from IEEE (Institute of Electrical and Electronics Engineers) (pp. 86 - 87) website: https://ieeexplore.ieee.org/document/8267589/

McKinney, W. (2017). Python for Data Analysis (2nd ed.). O'Reilly Media. ISBN: 9781491957660 https://www.oreilly.com/library/view/python-for-data/9781491957653/

McRorey, L. (2025) Python: The Ultimate Programming Language for Modern Applications https://teamstation.dev/nearshore-it-staffing-articles/insights?post=python-powerful-programming-language

Murel , J. & Kavlakoglu, E. (2024, March 25). Reinforcement learning. Retrieved from IBM website: https://www.ibm.com/topics/reinforcement-learn

Neumann, G. (2002) Programming Languages in Artificial Intelligence. In H. Bidgoli (ed.), *Encyclopedia of Information Systems*, (pp. 31-45). Academic Press, San Diego, CA.

https://www.sciencedirect.com/referencework/9780122272400/encyclopedia-of-information-systems

NTU Library. (2024). *Python for basic data analysis*. Nanyang Technological University. Retrieved March 24, 2025, from https://libguides.ntu.edu.sg/python/datatypes

Ogunsanya O. A. & Taiwo R. Enhancing Concrete Structures: Integrating Machine Learning and Deep Learning for Optimizing Materials Strength, Fire Resistance, and Impact Protection. *World Journal of Advanced Research and Reviews*. https://wjarr.com/content/enhancing-concrete-structures-integrating-machine-learning-and-deep-learning-optimizing

Oracle. (2022). AI vs Machine Learning. Retrieved from https://www.oracle.com/uk/artificial-intelligence/what-is-ai/ai-vs-machine-learning

Parlante, N. (2020). *Python function guide*. Stanford University. Retrieved March 24, 2025, from https://cs.stanford.edu/people/nick/py/python-function.html

Patnaik, A. (2023). *Python dictionary: A powerful tool for data engineering*. DZone. Retrieved March 24, 2025, from https://dzone.com/articles/python-dictionary-a-powerful-tool-for-data-engineer

PEP 206 – Python Advanced Library. (2000). Python.org. Retrieved November 19, 2024, from https://peps.python.org/pep-0206/

Perez, F., Granger, B. E.; Hunter, J. D. (2011). Python: An ecosystem for scientific computing. Computing in Science Engineering, 13(2), 13–21. https://doi.org/10.1109/mcse.2010.119

Pérez-Gil, Ó., Barea, R., López-Guillén, E., Bergasa, L. M., Gómez-Huélamo, C., Gutiérrez, R., & Díaz-Díaz, A. (2022). Deep reinforcement learning based control for Autonomous Vehicles in CARLA. Multimedia Tools and Applications, 81(3), 3553–3576. https://doi.org/10.1007/s11042-021-11437-3

Pozo Ramos, L. (2025). Python mutable vs immutable types: What's the difference? Real Python. Retrieved March 24, 2025, from https://realpython.com/python-mutable-vs-immutable-types/ing

Prabu, M., Sountharrajan, S., Suganya, E., & Bavirisetti, D. P. (2024). Contribution of Python to Improving Efficiency in Artificial Intelligence and Advancing Automation Capabilities. In Smart Computing Techniques in Industrial IoT (pp. 201-218). Singapore: Springer Nature Singapore.http://dx.doi.org/10.1007/978-981-97-7494-4_1

Pranav, K. V. R., & Sarma, K. J. (2023). Origin, development, and uses of machine learning. International Journal For Multidisciplinary Research, 5(1), 2. https://doi.org/10.36948/ijfmr.2023.v05i01.1367

Protasiewicz, J. (2024, September 2). Python vs. C # : A comprehensive guide for 2024. Retrieved from Netguru website: https://www.netguru.com/blog/python-vs-c-sharpPranav K. V. R. & Sarma K. J. (2023). Origin, Development and Uses of Machine Learning. *International Journal For Multidisciplinary Research*, *5*(1), 1367. https://doi.org/10.36948/ijfmr.2023.v05i01.1367

Rao, A. (2024). *Python Functions: A Complete Beginners Guide*. Edureka. Retrieved March 24, 2025, from https://www.edureka.co/blog/python-functions

Rahman, L. A., & Rana, M. E. (2021). The convergence between big data and the cloud: A review. 2021 International Conference on Data Analytics for Business and Industry. IEEE (pp 593-595). Retrieved from http://dx.doi.org/10.1109/icdabi53623.2021.9655772

Rohith, M. S. (2023). *Python inheritance common practices and pitfalls: Diamond problem, mixins, and others*. Towards AI. https://towardsai.net/p/l/python-inheritance-common-practices-and-pitfalls-diamond-problem-mixins-and-others (Retrieved April 12, 2025)

Rossum, Guido. Van. (January 2009). "The History of Python: A Brief Timeline of Python". *The History of Python*. Retrieved November 19, 2024.

Salehin, I., Islam, Md. S., Saha, P., Noman, S. M., Tuni, A., Hasan, Md. M., & Baten, Md. A. (2024). AutoML: A systematic review on automated machine learning with neural architecture search. Journal of Information and Intelligence, 2(1), 52–81. https://doi.org/10.1016/j.jiixd.2023.10.002

S, Premanand. (2024). Python multithreading: Concurrency and parallel execution in Python. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2023/08/exploring-multithreading-concurrency-and-parallel-execution-in-python/ (Retrieved April 12, 2025)

Severance, C. (2016). *Tuples are immutable*. Python for Everybody. Runestone Academy. Retrieved March 24, 2025, from https://runestone.academy/ns/books/published/py4e-int/tuples/tuplesareimmutable.html

Simplilearn. (2025). *Functions in Python | Definition, types and examples*. *Python Tutorial for Beginners* (Lesson 18 of 51). Simplilearn. Retrieved March 24, 2025, from https://www.simplilearn.com/tutorials/python-tutorial/python-functions

Singh, G., Misra, D., & Narang, P. (2022). Visualization and analysis of populations. *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)*. https://doi.org/10.1109/com-it-con54601.2022.9850886

Singh, V. (2024). *Types of functions in Python*. Shiksha. Retrieved March 24, 2025, from https://www.shiksha.com/online-courses/articles/types-of-functions-in-python/

Singh P. (2024). Python Operators: A Comprehensive Guide. Retrieved March 25, 2025, from https://www.analyticsvidhya.com/blog/2024/01/mastering-python-operators-a-comprehensive-guide/

Singh S., (2025). Comparative Analysis of Python's Role in AI and Machine Learning. International Journal of Research in Engineering, Science, and Management Volume 8, Issue 2. https://journal.ijresm.com/index.php/ijresm/article/view/3219/3278

S. Mihajlovic, A. Kupusinac, D. Ivetic, & I. Berkovic. (2020). Use of Python in the Field of Artificial Intelligence. Conference Paper. https://www.researchgate.net/publication/366578422_The_Use_of_Python_in_the_field_of_Artifical_Intelligence

"Status of Python Versions". (July 2024). *Python Developer's Guide*. Retrieved November 20, 2024, from https://devguide.python.org/versions/

Stephen M., W. I. (2022). What is Lisp (Programming Language)? — Klu. Retrieved from https://klu.ai/glossary/lisp-programming-language

StudySmarter. (2024, December 12). Loops in Programming. Retrieved from StudySmarter UK https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/loop-in-programming

Sutton, R. S., & Barto, A. G. (2018a). Reinforcement Learning, second edition: An Introduction. MIT Press (pp. 16-17).

Tas, S. (2024, February 21). Garbage collection and memory management in Python. Retrieved from Built In website: https://builtin.com/articles/garbage-collection-in-python

Toyosi, M. F., Jaiyeoba, G., Oluwafemi, T. O., & Muhideen, O. O. (2024). The Effect of Smart Factory on the Continuous Improvement of the Production Process: A Review. International Journal of Engineering and Modern Technology, 10(1), 83–107. https://doi.org/10.56201/ijemt.v10.no1.2024.pg83.107

Turley, D. (2024, May 29). The low-code/no-code revolution: Democratizing development and reshaping software engineering. The AI Journal. Retrieved from https://aijourn.com/the-low-code-no-code-revolution-democratizing-developMent-and-reshaping-software-engineering/

Turner, R. (2012, October 24). Literate Programming in Lisp - Computer Science - University of Maine. Retrieved September 2, 2024, from Computer Science website: https://umaine.edu/cs/research/features/literate-programming-in-lisp/

Van Rossum, G. (August 2000). "SETL (was: Lukewarm about range literals)". *Python-Dev* (Mailing list). Retrieved November 20, 2024.

Venners, B. (January 2003). "The Making of Python". *Artima Developer*. Artima. Retrieved November 20, 2024, from https://www.artima.com/articles/the-making-of-python

Vidvan, T. (2020, May 13). Top 20 Python deep learning applications you must know. Retrieved September 2, 2024, from TechVidvan website: https://techvidvan.com/tutorials/deep-learning-applications/

Weerts, H., Dudík, M., Edgar, R., & Madaio, M. (2023, March 29). Fairlearn: Assessing and improving fairness of AI systems. 2-3

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., & Rush, A. (2020). *Transformers: State-of-the-Art Natural Language Processing*. In Q. Liu & D. Schlangen (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 38–45). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-demos.6

Worsley S. (July 2024). What is Python? Everything You Need to Know to Get Started. *DATACAMP*. Retrieved August 13, 2024, from https://www.datacamp.com/blog/all-about-python-the-most-versatile-programming-language

Ziller, A., Trask, A., Lopardo, A., Szymkow, B., Wagner, B., Bluemke, E., … Kaissis, G. (2021). PySyft: A library for easy federated learning. In Studies in Computational Intelligence (pp. 111–139). Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-030-70604-3_5