Math and Machine - The Theoretical Python

Mostafizur Rahaman, Lecturer, Dept. of ESE, KUET

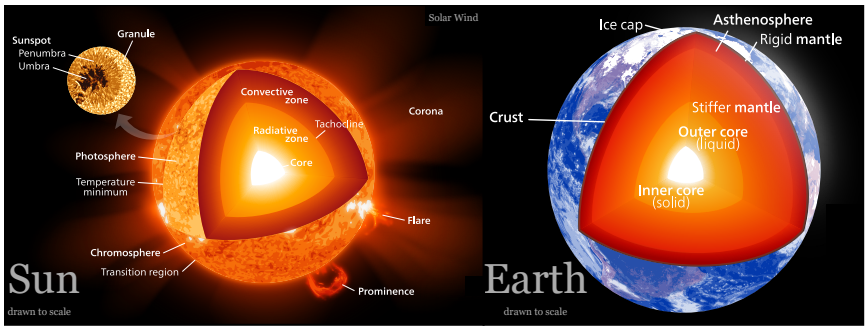
September 25, 2024

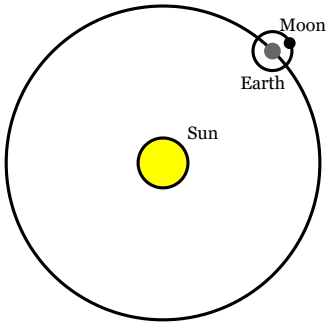
Contents

1 | The Machine - Hardware & Software

Hardware is the body, software is the spirit

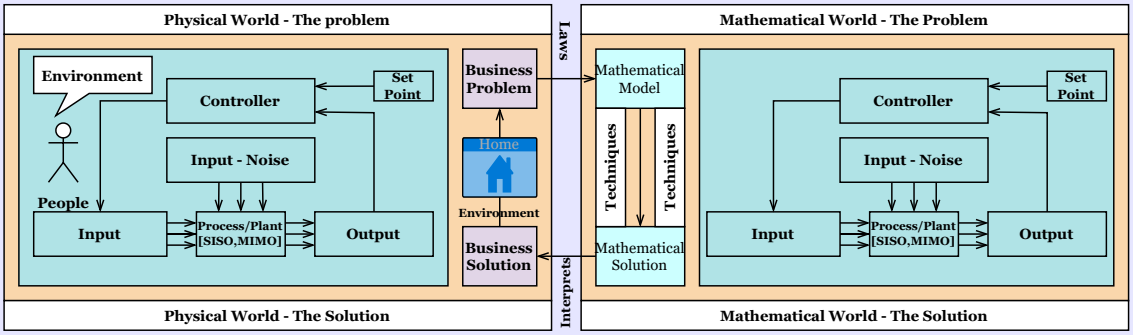
In nature, the Sun, Earth, and Moon constitute the solar system. The Sun emits light and heat, while Earth orbits around it, providing a habitable environment. Additionally, Earth's natural satellite, the Moon, orbits around Earth, influencing tides and climate stability.



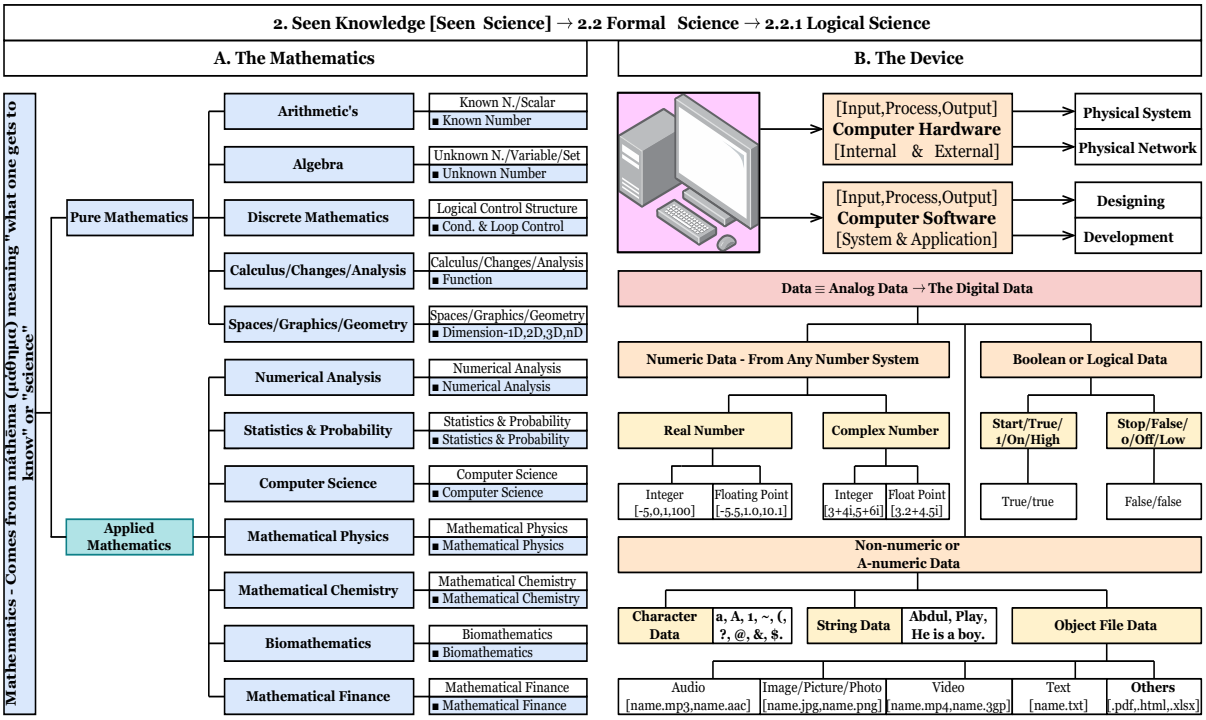


Definition of Science

Science: Science from the Latin *scientia*, meaning "knowledge" is defined as an organized body of knowing about the natural universe and the processes by which that knowing is acquired and tested. **Science** can be broadly classified into two categories: **seen science** and **unseen science**. Seen science involves the study of observable phenomena using the five senses or scientific instruments, while unseen science explores concepts beyond direct observation through mathematical models and theoretical frameworks.



Seen Science can be broadly categorized into two main branches: **empirical science** and **formal science**. Empirical science is based on observation and experimentation. The natural sciences are empirical, and sometimes the terms are used interchangeably. Formal science is based on theoretical symbols, rules, and logic. Mathematics is a formal science.



Formal/Logical Science studies abstract structures using formal systems, including logic, mathematics, and computer science. Unlike natural and social sciences, which rely on empirical methods, formal sciences use language tools to characterize abstract structures. They help to describe the physical world, assisting other sciences in making inferences. Logical science is broadly divided into two branches

A. Mathematics: "The Mathematics" provides the tools and language necessary to describe and analyze phenomena in various fields such as physics, engineering, economics, and biology. It forms the basis for problem-solving and decision-making in both theoretical and applied contexts.

B. Devices: "The Devices" provide the practical platform for the application of mathematical knowledge across a wide range of fields and disciplines. They enable the transformation of abstract mathematical theories into tangible solutions that address real-world problems and challenges. Indeed, Computer a special type of device, which combines both hardware and software components, plays a crucial role in facilitating the application of mathematical knowledge across various fields and disciplines.

Hardware-Related Jobs (Computer Hardware): The computer hardware includes components such as the central processing unit (CPU), memory (RAM), storage devices (hard drives, solid-state drives), input/output devices (keyboard, mouse, monitor). These hardware-related jobs are broadly categorised into

a. Physical System: Involves designing, building, and maintaining physical computer systems, including hardware components such as CPUs, memory modules, and input/output devices. This encompasses tasks related to assembling, configuring, and troubleshooting computer hardware.

b. Physical Network: Focuses on designing, implementing, and managing physical network infrastructure, including routers, switches, cables, and other networking hardware. This includes tasks related to network architecture, installation, maintenance, and optimization.

Software-Related Jobs (Computer Software): System Software: This includes operating systems, device drivers, and utility programs that manage and control the computer hardware and provide a platform for running application software. Application Software: These are programs designed for specific tasks or purposes, such as word processors, web browsers, games, and multimedia applications. These software-related jobs are broadly categorised into

a. Designing and Simulation: This involves using prebuilt software tools to design and simulate systems or processes. For example, engineers might use computer-aided design (CAD) software to design buildings or mechanical parts, or simulation software to model complex systems.

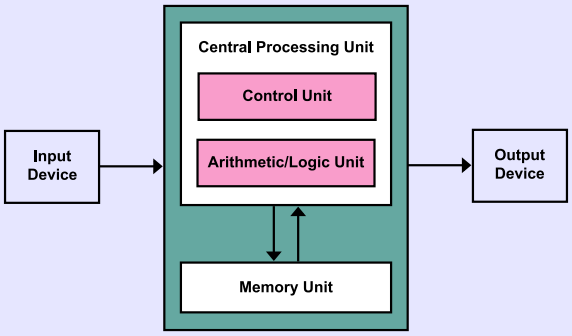
b. Programming and Development: This encompasses the creation of software applications and systems through programming languages and development tools. Programmers write code to create software applications, ranging from mobile apps to enterprise-level software systems, while developers handle the broader process of software development, including planning, coding, testing, and deployment.

1.1 Machine - The Computer Hardware

1.1.1 Computer - The Definition and Types

Computer - The Definition as Hardware

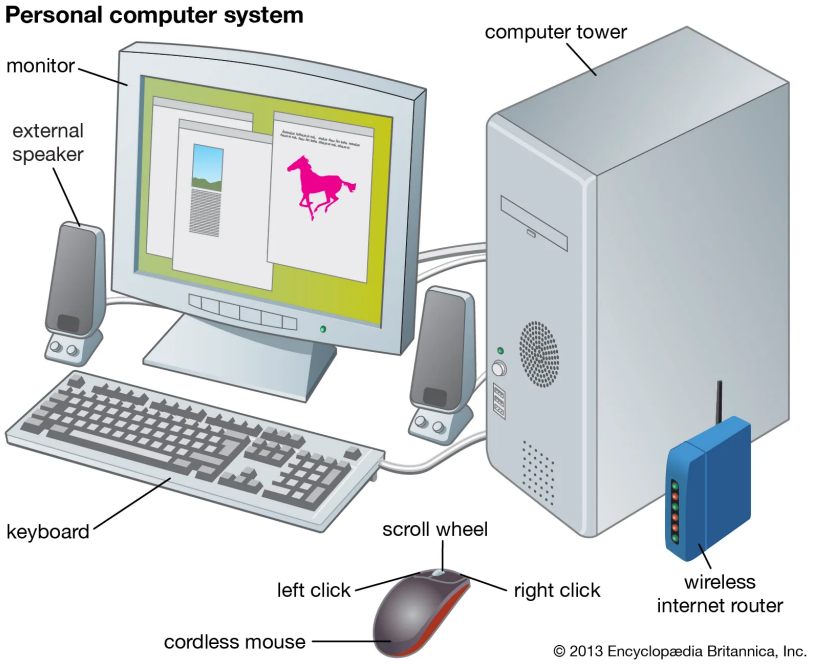
A computer is a programmable machine – something that can execute a sequence of instructions to complete a task. or A computer is a machine that can be instructed to carry out sequences of arithmetic or logical operations automatically via computer programming. The British inventor Charles Babbage conceived the mechanical computer, the difference Engine in 1822 and later, in 1842, proposed the design for the Analytical Engine. The Analytical Engine featured a memory unit and utilized card input/output mechanisms for data and instructions. Von Neumann. Scientific genius and consultant on the ENIAC project, contributed to the development of the Electronic Discrete Variable Automatic Computer (EDVAC) in 1950.



1.1.2 Computer - The Why We Need

Category	Human	Computer
Visual Perception	Eyes and Brain Processing	Cameras and Image Processing Algorithms
Auditory Perception	Ears and Auditory System	Microphones and Audio Processing Algorithms
Cognitive Processing	Heart and Brain	Central Processing Unit (CPU)
Verbal Output	Vocal Cords and Mouth	Speaker and Text-to-Speech Synthesis
Physical Movement	Muscles and Skeletal System	Automation and Robotics

1.1.3 Computer - The How It Works

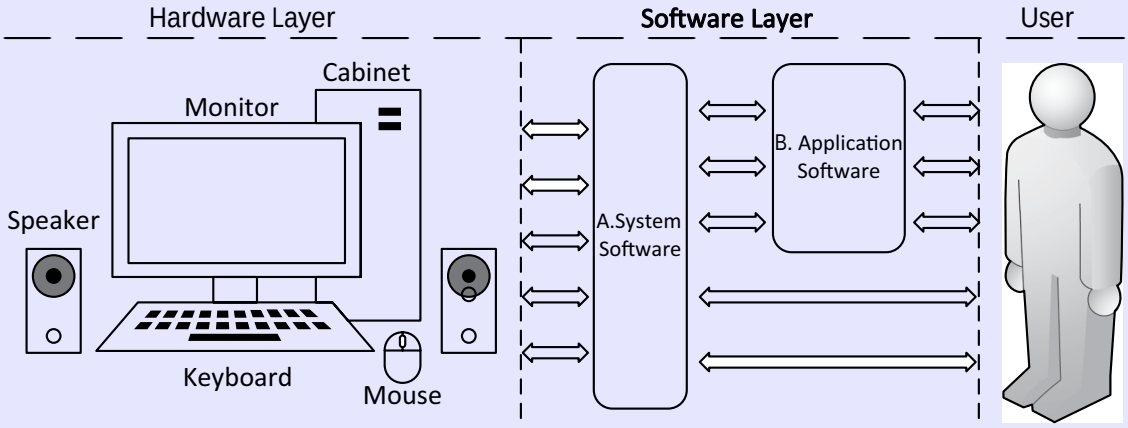


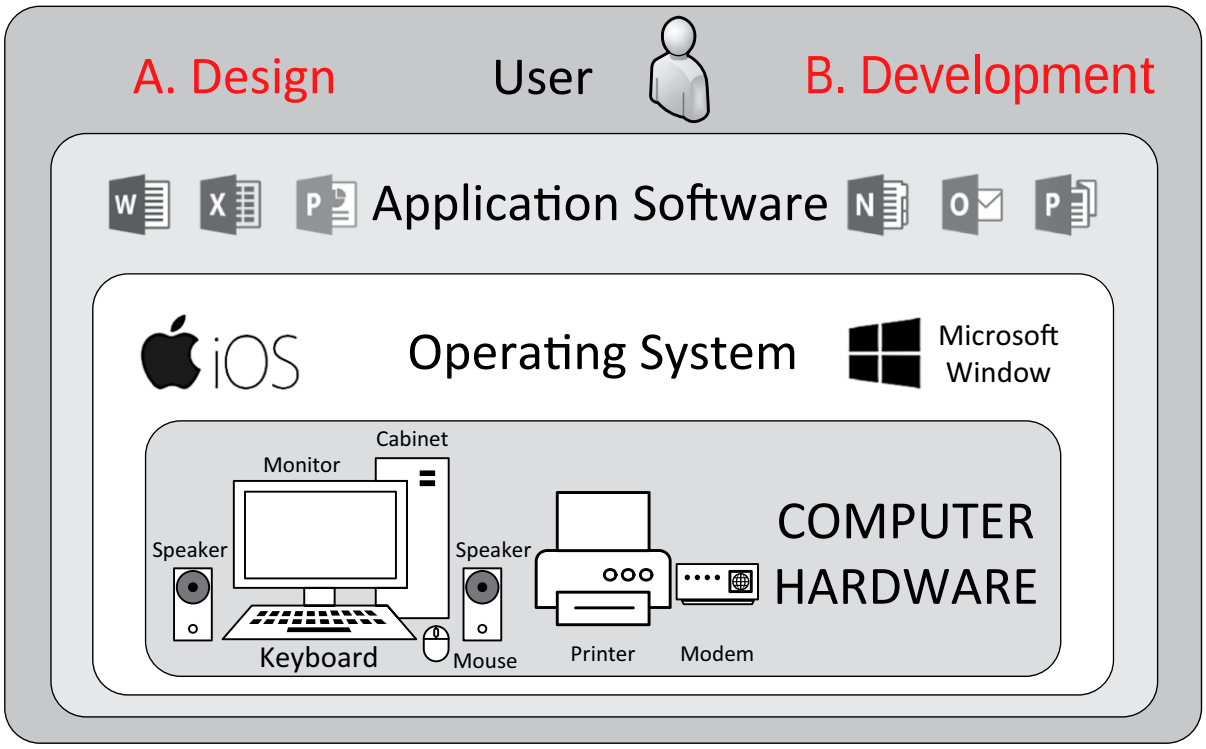
1.2 Machine - The Computer Software

1.2.1 Program - The Definition and Types

Program - The Definition ≡ Stored + Written Instructions + Computer Language + OS + Perform Task

A computer program is a stored sequence of written instructions using computer language for a computer to perform a task. We will also call this code and software. The process of writing a program is called programming or coding. The first computer program is generally dated to 1843, when mathematician Ada Lovelace published an algorithm to calculate a sequence of Bernoulli numbers, intended to be carried out by Charles Babbage's Analytical Engine. One significant milestone in the history of digital programming is the development of the Electronic Numerical Integrator and Computer (ENIAC) during World War II in 1947. One of the first computers to fully embody the concept of the stored-program computer was the Electronic Discrete Variable Automatic Computer (EDVAC), proposed by John von Neumann in the late 1940s.





1.2.2 Program - The How It Works

Python - The Defⁿ ≡ Open Source + General Purpose + High Level + Easy Syntax + Dynamic Semantics

Python is an open-source interpreted general-purpose (or multi-purpose) high-level programming language with easy syntax and dynamic semantics. Python was developed by Guido van Rossum in 1991 at the Corporation for National Research Initiatives (CNRI) in the United States and was named after the BBC TV show Monty Python's Flying Circus. Example: Python - `https://www.python.org/downloads/`.

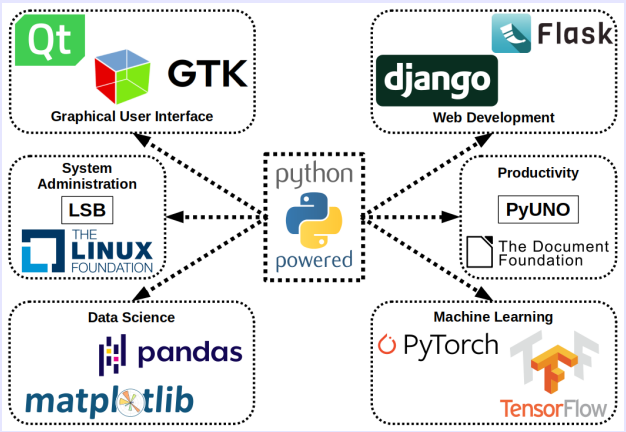
Python - The IDE/Environment/Desktop ≡ GUI + Editor + Debugger + Translator + More Optional'

An Integrated Development Environment (IDE) is a Graphical User Interface (GUI) based application software for program, code, or software development, which includes a source code editor for writing and modifying code, a debugger for identifying and resolving errors within the code, and a translator or compiler for converting code into executable programs. Example: VSCode - `https://code.visualstudio.com/download`, Anaconda - `https://www.anaconda.com/download`.

Python - The Translator ≡ Predefined Program + Human Convenient Language ⟶ Machine Language

A translator or programming language processor is a predefined computer program that converts the programming instructions written in human convenient form into machine language codes that the computers understand and process.

Python - The Why



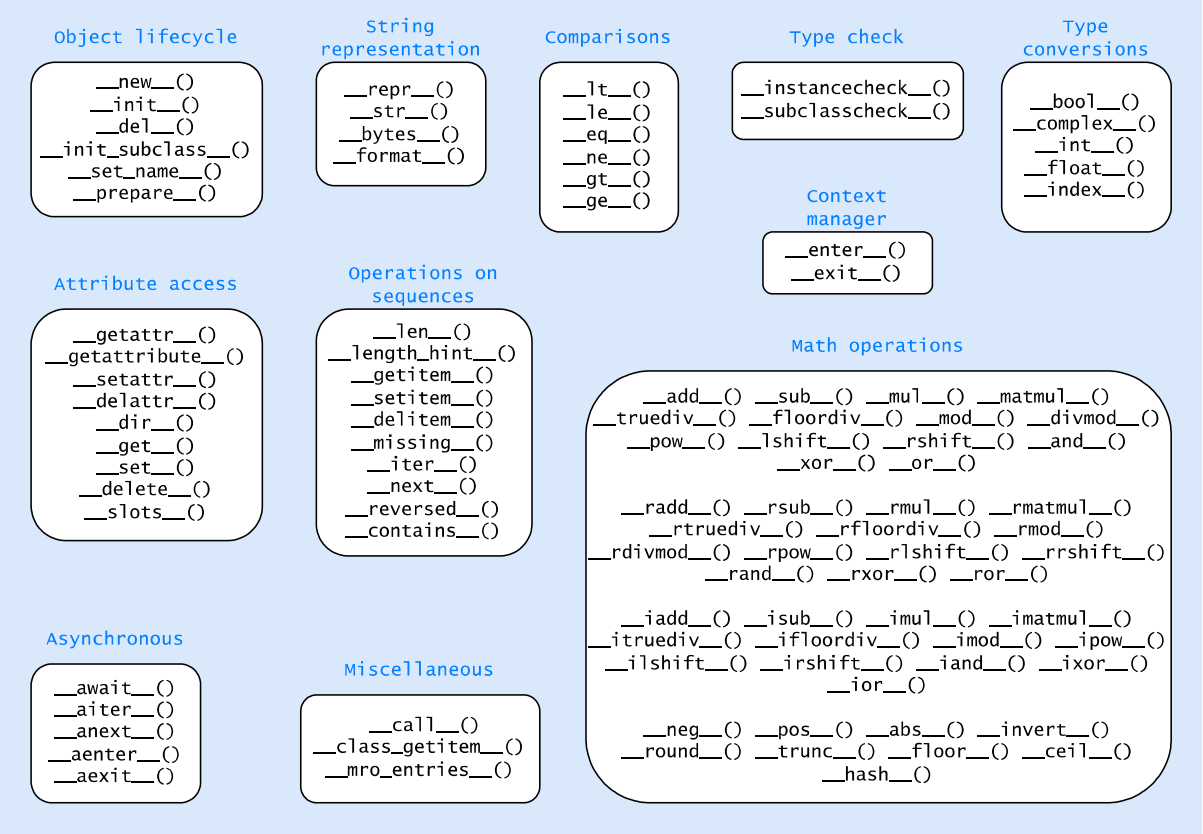
Python - The How

1. Download the Application Software - File in the name 'Python.exe'
2. Install the Application Software - Path in the environment variable: In both the Command Prompt (cmd) and PowerShell, users can interact with the operating system through a command-line interface. When someone install software, 'environment variables' may be added or modified to include the necessary paths for the software to function correctly. This is particularly common for software that requires executable files or libraries to be accessible from the command line or other applications.
3. Download the IDE for Application Software - 'VSCode' or 'Anaconda': Both Visual Studio Code and Anaconda are popular choices for software development and data science. Visual Studio Code is a lightweight and versatile code editor, while Anaconda is a distribution of Python and R programming languages for scientific computing and data science. Depending on your needs, you can choose the one that best suits your requirements.
4. Install the Extensions and Manage the Application Software - 'VSCode' or 'Anaconda': One can easily install extensions in Visual Studio Code and manage software environments and packages in Anaconda Navigator according to your development or data science needs.
5. One Need to Learn keyword - 'Python': import builtins; print(dir(builtins)); import keyword; print(keyword.kwlist);

Builtins Functions: ~~'ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BaseExceptionGroup', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EncodingWarning', 'EnvironmentError', 'Exception', 'ExceptionGroup', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError'~~. '__IPYTHON__', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'aiter', 'all', 'anext', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate', 'eval', 'exec', 'execfile', 'filter', 'float', 'format', 'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range', 'repr', 'reversed', 'round', 'runfile', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip',

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

Class Methods:



2 | The Expression - Data & Operations

Expressions create and process objects (Data)

2.1 Expression - The Data

2.1.1 Data - The Definition and Types

Datum to Data: Data is simply any numbers, letters or symbols that can be entered into a computer system, as a raw unorganized facts that needs to be processed that can be something simple and seemingly random and useless until it is organized by a process. Data file or file (file_name.file_extension): A computer file is set of data or information which is stored on a storage partition using a unique name to identify it. Data is classified as listed below: Rule for 'The Data' in Python is - Data_Name = Data_Value or Variable_Name = Data_Type.

Data Name or Variable Name: The variable name is the name of the location of memory where the data value is stored. A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

A. A variable name must start with a letter or the underscore character and cannot start with a number

B. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

C. Variable names are case-sensitive (age, Age, and AGE are three different variables) and can be Python keywords or built-in function names, but it's not recommended.

Data Types		Definition	Properties	Example
Basic Types	NoneType int float complex bool	A null value or absence of a value whole numbers, +ve or -ve, no decimal points decimal numbers, positive or negative True or False values constituents are real and imaginary parts	- - - - -	a = None b1=1234, b2=0b111, b3=00111, b4=0x111 c = 3.1415 d = 3+4j e1 = True, e2 = False
Collection Types	str ~~bytes~~ ~~bytearray~~ ~~memoryview~~	characters enclosed in any type of quotation mark	Indexed, Ordered, Not Changeable, Duplicates	m1 = 'Hello World', m2 = "World's Mine"
	tuple ~~range~~	collection of elements, comma seperated within ()	Indexed, Ordered, Not Changeable, Duplicates	n1 = ("apple", "banana", "cherry") n2 = tuple(("apple", "banana", "cherry"))
	set ~~frozenset~~	collection of elements, comma seperated within {}	No Indexed, No Ordered, Not Changeable, No Duplicates	o1 = {"apple", "banana", "cherry"} o2 = set(("apple", "banana", "cherry"))
	list	collection of elements, comma seperated within []	Indexed, Ordered, Changeable, Duplicates	p1 = ["apple", "banana", "cherry"] p2 = list(("apple", "banana", "cherry"))
	dict	collection of key-values, comma seperated in {}	No Indexed, Ordered, Changeable, No Duplicates	q1 = {"name" : "John", "age" : 36} q2 = dict(name="John", age=36)
User Defined Types	File Function Class	File objects used for file input / output operations take inputs, execute statements, and return value object attributes(variables) & methods(functions)	- - -	x = open('filename.mp3', 'rb') y = lambda a,b,c: a*b**c class myclass: z = 5

Data Values or Data types : In Python Data Values are generally classified into three broad categories. These categories provide a structured way to understand and organize the various data types available in Python, catering to different needs and scenarios in programming:

A. Fundamental Data Types : These are basic data types provided by Python to represent simple values.

- **None Type:** Represents a null value or absence of a value.
- **Integer:** Represents whole numbers, positive or negative, without decimal points.
- **Float:** Represents decimal numbers, positive or negative.
- **Complex:** Represents complex numbers with real and imaginary parts.
- **Boolean:** Represents True or False values.

B. Collection Data Types : These data types are used to store collections of values.

- **String:** Represents sequences of characters enclosed in quotation marks.
- **Tuple:** Ordered and unchangeable collection of elements, allowing duplicate members.
- **Set:** Unordered, unchangeable, and unindexed collection of unique elements.
- **List:** Ordered and changeable collection of elements, allowing duplicate members.
- **Dictionary:** Ordered collection of key-value pairs, where keys are unique.

C. User-Defined Data Types : These are data types defined by the user such as classes and functions.

- **File:** Represents file objects used for file input/output operations.
- **Function:** Represents callable objects created using the `def` keyword to define reusable blocks of code.
- **Class:** Represents blueprints for creating objects with specific properties and methods.

2.1.2 Data - The Call and Index/Slice

Variable Calling Syntax ≡ variable_name, & **Variable Indexing/Slicing Syntax** ≡ variable_name[start:end:step]

* **start**: the index of the first element to include in the slice. **end**: the index of the first element to exclude from the slice (i.e. the slice will include all elements up to, but not including, the element at this index). **step**: the increment between the elements to include in the slice (default is 1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
variable =	R	e	a	d		t	h	e		b	o	o	k	.
	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

word = “amazing”

A. word[0:7]: ’amazing’ (indices 0 to 6). This slice starts at index 0 and ends just before index 7, capturing the entire word ‘amazing’.

B. word[2:5]: ’azi’ (indices 2 to 4). This slice starts at index 2 and ends before index 5, resulting in ‘azi’.

C. word[-7:-3]: ’amaz’ (indices -7 to -4). Negative indices count from the end of the string. This slice gets ‘amaz’ starting 7 places from the end up to, but not including, 3 places from the end.

D. word[-5:-1]: ’azin’ (indices -5 to -2). Similar to the previous, but starts 5 places from the end and goes up to the second last character, giving ‘azin’.

E. word[1:6:2]: ’mzn’ (every 2nd character from index 1 to 5). The additional step argument here means every second character between indices 1 and 5 is taken, resulting in ‘mzn’.

F. word[-7:-3:3]: ’az’ (every 3rd character from -7 to -4).This takes every third character between the 7th from the end and the 4th from the end, resulting in ‘az’.

G. word[::-2]: ’giaa’ (every 2nd character in reverse).A negative step causes the string to be traversed backwards. Every second character from the end is ‘giaa’.

H. word[::-1]: ’gnizama’ (entire string reversed). This selects every character starting from the end and moving towards the beginning of the string, effectively reversing the entire string.

2.1.3 Data - The Methods and Actions

A. **class NoneType Methods List:** -

B. **class int Methods List:** as_integer_ratio, bit_count, bit_length, conjugate, denominator, from_bytes, imag, numerator, real, to_bytes,

C. **class float Methods List:** as_integer_ratio, conjugate, fromhex, hex, imag, is_integer, real,

D. **class complex Methods List:** conjugate, imag, real,

E. **class bool Methods List:** as_integer_ratio, bit_count, bit_length, conjugate, denominator, from_bytes, imag, numerator, real, to_bytes,

F. **class str Methods List:** capitalize, casefold, center, count, encode, endswith, expandtabs, find, format, format_map, index, isalnum, isalpha, isascii, isdecimal, isdigit, isidentifier, islower, isnumeric, isprintable, isspace, istitle, isupper, join, ljust, lower, lstrip, maketrans, partition, removeprefix, removesuffix, replace, rfind, rindex, rjust, rpartition, rsplit, rstrip, split, splitlines, startswith, strip, swapcase, title, translate, upper, zfill.

Escape	Meaning
\newline	Ignored (continuation line)
\\	Backslash (stores one \)
\', \"	Single quote (stores '), Double quote (stores ")
\a	Bell
\b	Backspace
\f	Formfeed
\n	Newline (linefeed)
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\xhh	Character with hex value hh (exactly 2 digits)
\ooo	Character with octal value ooo (up to 3 digits)
\0	Null: binary 0 character (doesn't end string)
\Nid	Unicode database ID
\uhhhh	Unicode character with 16-bit hex value
\Uhhhhhhhh	Unicode character with 32-bit hex value
\other	Not an escape (keeps both \and other)

Code	Meaning
s	String (or any object's str(X) string)
r	Same as s, but uses repr, not str
c	Character (int or str)
d	Decimal (base-10 integer)
i	Integer
u	Same as d (obsolete: no longer unsigned)
o	Octal integer (base 8)
x	Hex integer (base 16)
X	Same as x, but with uppercase letters
e, E	Floating point with exponent lowercase, Same as e but uses uppercase letters
f, F	Floating-point decimal, Same as f but uses uppercase letters
g	Floating-point e or f
G	Floating-point E or F
%	Literal % (coded as %%)

G. **class tuple Methods List:** count, index,

H. **class set Methods List:** add, clear, copy, difference, difference_update, discard, intersection, intersection_update, isdisjoint, issubset, issuperset, pop, remove, symmetric_difference, symmetric_difference_update, union, update,

I. **class list Methods List:** append, clear, copy, count, extend, index, insert, pop, remove, reverse, sort,

J. **class dict Methods List:** clear, copy, fromkeys, get, items, keys, pop, popitem, setdefault, update, values,

2.2 Expression - The Operations

Operator: An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. They usually form a part of mathematical or logical expressions.

Format of an Expression ≡	Operand/Variable	Operator/Sign	Operand/Variable	⇒ Result/Data
*** Example of the Format ≡**	100	+	200	⇒ 300

Types of operator according to the operand: Python divides the operators into the following groups -

- ▷ Unary operator: −25
- ▷ Binary operators: 12 + 14

Types of Operators according to work: Python divides the operators into the following groups -

- ▷ **Assignment operators:** Used to assign values to variables. Examples include '=', '+=', '-=' etc.
- ▷ **Arithmetic operators:** Used to perform mathematical operations on numeric operands. Examples include '+' (addition), '-' (subtraction), '*' (multiplication), '/' (division), '**' (exponentiation), and '//' (floor division).
- ▷ **Relational/Comparison operators:** Used to compare two values. Examples include '==' (equal to), '!=' (not equal to), '<' (less than), '>' (greater than), '<=' (less than or equal to), and '>=' (greater than or equal to).
- ▷ **Logical operators:** Used to combine conditional statements. Examples include 'and' (logical AND), 'or' (logical OR), and 'not' (logical NOT).
- ▷ **Membership operators:** Used to test if a value is present in a sequence or not. Examples include 'in' (present in) and 'not in' (not present in).
- ▷ **Identity operators:** Used to compare the memory locations of two objects. Examples include 'is' (is the same object) and 'is not' (is not the same object).
- ▷ **Bitwise operators:** Used to perform bit-level operations on binary numbers. Examples include '&' (bitwise AND), '|' (bitwise OR), '^' (bitwise XOR), '~' (bitwise NOT), '«' (left shift), and '»' (right shift).

2.2.1 Operations - The Elementary Operators

Assignment operators: Used to assign values to variables. It can be simple and multiple.

Operator	Example	Same As
=	x=5; y='Read'; x,y ='Read','Book'; x=y='Read'; a,b,c,d='Read', a,*b ='Read'	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
»=	x »= 3	x = x »3
«=	x «= 3	x = x «3

Arithmetic operators: Used to perform mathematical operations on numeric operands.

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Relational/Comparison operators: Used to compare two values.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators: Used to combine conditional statements.

Operator	Description	Example
and	Returns True if both statements are true	True and False
or	Returns True if one of the statements is true	True or False
not	Reverse the result, returns False if the result is true	not True

Membership operators: Used to test if a value is present in a sequence or not.

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Identity operators: Used to compare the memory locations of two objects.

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Bitwise operators: Used to perform bit-level operations on binary numbers.

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

2.2.2 Operations - The Compound Operators

Compound operation in Python refers to performing multiple operations in a single line of code. This is achieved through the use of operator precedence and grouping with parentheses. Operations are performed left-to-right, meaning that when an expression contains multiple operators, they are applied in the order they appear, from left to right as PEMDASRL.

Order	Operator	Description
0	(expressions...), [expressions...], {key: value...}, {expressions...}	Binding or parenthesized expression, list display, dictionary display, set display
1	x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
2	await x	Await expression
3	**	Exponentiation
4	+x, -x, ~x	Positive, negative, bitwise NOT
5	*, @, /, //, %	Multiplication, matrix multiplication, division, floor division, remainder
6	+, -	Addition and subtraction
7	«, »	Shifts
8	&	Bitwise AND
9	^	Bitwise XOR
10	|	Bitwise OR
11	in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
12	not x	Boolean NOT
13	and	Boolean AND
14	or	Boolean OR
15	if – else	Conditional expression
16	lambda	Lambda expression
17	:=	Assignment expression

Example of Mathematical Simplification:

A. $3 \times 4 + 2 \times 5$ Simplification: $12 + 10 = 22$

B. $(4 + 3) \times 2 - 5$ Simplification: $7 \times 2 - 5 = 9$

C. $2^3 + 5\%2 \times 4$ Simplification: $8 + 1 \times 4 = 12$

D. $(6 - 3) \times \left(\frac{8}{4}\right) + 2$ Simplification: $3 \times 2 + 2 = 8$

E. $4 \times (2 + 3)^2$ Simplification: $4 \times 25 = 100$

F. $7 - 3 \times 2^2 + 5$ Simplification: $7 - 12 + 5 = 0$

G. $(4 + 2) \times \left(\frac{10}{5} - 1\right)$ Simplification: $6 \times (2 - 1) = 6$

```
result_A      = 3 * 4 + 2 * 5                    # Expression A  Ans:  22
alt_result_A  = eval("3 * 4 + 2 * 5")            # Alternate way to solve
result_B      = (4 + 3) * 2 - 5                  # Expression B  Ans:   9
alt_result_B  = eval("(4 + 3) * 2 - 5")          # Alternate way to solve
result_C      = 23 + 5 % 2 * 4                   # Expression C  Ans:  12
alt_result_C  = eval("23 + 5 % 2 * 4")           # Alternate way to solve
result_D      = (6 - 3) * (8 / 4) + 2            # Expression D  Ans:   8
alt_result_D  = eval("(6 - 3) * (8 / 4) + 2")    # Alternate way to solve
result_E      = 4 * (2 + 3)**2                   # Expression E  Ans: 100
alt_result_E  = eval("4 * (2 + 3)**2")           # Alternate way to solve
result_F      = 7 - 3 * 2**2 + 5                 # Expression F  Ans:   0
alt_result_F  = eval("7 - 3 * 2**2 + 5")         # Alternate way to solve
result_G      = (4 + 2) * (10 / (5 - 1))         # Expression G  Ans:   6
alt_result_G  = eval("(4 + 2) * (10 / (5 - 1))") # Alternate way to solve
```

P1.1.1 - Creation of Different Data Types: Explain how one would create various data types in Python:

A. NoneType Instance: Create an instance representing the NoneType.

B. Integer Instance: Create an instance representing the integer 42.

C. Float Instance: Create an instance representing the float 3.14.

D. Complex Instance: Create an instance representing a complex number, for example, 2 + 3i.

E. Boolean Instances: Create two instances representing the boolean values True and False.

F. String Instance: Create an instance representing the string “Hello, world!”.

G. Tuple Instance: Create an instance representing a tuple containing the elements “apple”, “banana”, and “cherry”

H. Set Instance: Create an instance representing a set containing elements 1, 2, 3, 4, and “Read the Book”

I. List Instance: Create an instance representing a list containing the elements 1, None, 3, 4, and ‘Do Good‘

J. Dictionary Instance: Create an instance representing a dictionary with the keys “name” and “age”, and their corresponding values “John” and 30.

K. Create a string named `my_string` that contains the following sentences:

```
"Read in the name of your Lord who created.
Created man from a clinging substance.
Read, and your Lord is the most Generous.
Who taught by the pen.
Taught man that which he knew not."
```

L. Create a tuple named `my_tuple` that contains the following elements: Integer: 42, Float: 3.14, String: “Hello”, Boolean: True, List: [1, 2, 3], Tuple: (“apple”, “banana”), Dictionary: {“name”: “John”, “age”: 30}, Set: 1, 2, 3

M. Create a set named `my_set` that contains the following elements: Integer: 42 , Float: 3.14 , String: “Hello” , Boolean: True , Tuple: (1, 2, 3) , String: “apple” (Note: Duplicates are automatically removed in a set) ,

N. Create a list named `my_list` that contains the following elements: Integer: 42 , Float: 3.14 , String: “Hello” , Boolean: True , List: [1, 2, 3] , Tuple: (“apple”, “banana”) , Dictionary: {“name”: “John”, “age”: 30} , Set: {1, 2, 3} ,

O. Create a dictionary named `my_dict` that contains the following key-value pairs: “integer”: 42 , “float”: 3.14 , “string”: “Hello” , “boolean”: True , “list”: [1, 2, 3] , “tuple”: (“apple”, “banana”) , “nested_dict”: {“name”: “John”, “age”: 30} , “set”: {1, 2, 3}

P1.1.2 - Indexing and Slicing of Data Types: How one would index and slice data types in Python:

A. Given the string “Hello, world!”, perform the following tasks:

- **a.** Retrieve the first character of the string.
- **b.** Retrieve the last character of the string.
- **c.** Retrieve the substring “Hello”.
- **d.** Retrieve the substring “world”.
- **e.** Reverse the string.

B. Given the string “The quick brown fox jumps over the lazy dog”, perform the following tasks:

- **a.** Retrieve the characters at even indices.
- **b.** Retrieve the characters at odd indices.
- **c.** Retrieve every third character starting from the second character.

d. Reverse the string using slicing.

C. Given the tuple (10, 20, 30, 40, 50), perform the following tasks:

a. Retrieve the first element of the tuple.
b. Retrieve the last element of the tuple.
c. Retrieve the sublist (20, 30).
d. Retrieve the sublist (30, 40, 50).
e. Reverse the tuple.

D. Given the tuple (5, 10, 15, 20, 25, 30, 35), perform the following tasks:

a. Retrieve elements at even indices.
b. Retrieve elements at odd indices.
c. Retrieve every third element starting from the second element.
d. Reverse the tuple using slicing.

E. Given the list [1, 2, 3, 4, 5], perform the following tasks:

a. Retrieve the first element of the list.
b. Retrieve the last element of the list.
c. Retrieve the sublist [2, 3].
d. Retrieve the sublist [3, 4, 5].
e. Reverse the list.

F. Given the list [10, 20, 30, 40, 50, 60, 70], perform the following tasks:

a. Retrieve elements at even indices.
b. Retrieve elements at odd indices.
c. Retrieve every third element starting from the second element.
d. Reverse the list using slicing.

G. Given the dictionary {”name”: “John”, “age”: 30, “city”: “New York”}, perform the following tasks:

a. Retrieve the value associated with the key “name”.
b. Retrieve the value associated with the key “age”.
c. Retrieve the value associated with the key “city”.

H. Given the dictionary {”apple”: 3, “banana”: 6, “cherry”: 9, “date”: 12}, perform the following tasks:

a. Retrieve keys at even indices.
b. Retrieve keys at odd indices.
c. Retrieve every third key starting from the second key.
d. Reverse the dictionary (swap keys with values).

P1.1.3 - Methods of each Data Type : How one would apply various methods in each data type.

String Operations: Concatenate the strings “Hello” and “World”.

A. Repeat the string “Python” three times.

B. Extract the first three characters from the string “Programming”.

C. Convert the string “HELLO” to lowercase.

D. Check if the string “Python” contains the substring “th”.

Tuple Operations: Create a tuple containing the numbers 1, 2, and 3.

A. Concatenate the tuple (4, 5, 6) with the previous tuple.

B. Find the index of the number 3 in the concatenated tuple.

Set Operations: Create a set containing the numbers 1, 2, and 3.

A. Add the number 4 to the set.

B. Remove the number 2 from the set.

C. Check if the set 1, 2, 3 is a subset of 1, 2, 3, 4, 5.

D. Find the intersection of the sets 1, 2, 3 and 3, 4, 5.

List Operations: Create a list containing the numbers from 1 to 5.

A. Append the number 6 to the list.

B. Remove the number 3 from the list.

C. Insert the number 10 at index 2.

D. Sort the list in ascending order.

Dictionary Operations: Create a dictionary with the keys "name" and "age" and their corresponding values.

A. Add a new key-value pair "city" with the value "New York".

B. Update the value of the "age" key to 25.

C. Remove the key "city" from the dictionary.

D. Check if the key "name" exists in the dictionary.

P1.2.1 - Elementary Operation: Solve problems on the elementary operation.

A. Addition: Write a Python program that adds two numbers and prints the result.

B. Subtraction: Write a Python program that subtracts one number from another and prints the result.

C. Multiplication: Write a Python program that multiplies two numbers and prints the result.

D. Division: Write a Python program that divides one number by another and prints the result.

E. Exponentiation: Write a Python program that raises a number to the power of another number and prints the result.

F. Square Root: Write a Python program that calculates the square root of a given number and prints the result.

G. Absolute Value: Write a Python program that finds the absolute value of a number and prints the result.

H. Integer Division: Write a Python program that performs integer division of one number by another and prints the result.

I. Modulus: Write a Python program that finds the remainder when one number is divided by another and prints the result.

J. Increment and Decrement: Write a Python program that increments a variable by 1 and decrements another variable by 1, then prints both variables.

K. Comparison Operators: Write a Python program that compares two numbers using comparison operators (`==`, `!=`, `<`, `>`, `<=`, `>=`) and prints the result.

L. Logical Operators: Write a Python program that performs logical operations (`and`, `or`, `not`) on two boolean variables and prints the result.

M. Bitwise Operators: Write a Python program that performs bitwise operations (`&`, `|`, `^`, `<<`, `>>`) on two integers and prints the result.

N. Assignment Operators: Write a Python program that uses all assignment operators (`=`, `+=`, `-=`, `*=`, `/=`, `%=`, `//=`, `**=`) with variables and prints their values.

O. String Concatenation: Write a Python program that concatenates two strings using the concatenation operator (`+`) and prints the result.

P1.2.2 - Compound Operation: Solve problems on the compound operators - https://mathforengineers.com/

1. **Mechanical Engineering**: Calculate the stress (σ) on a beam with a force (F) of 5000 N applied over an area (A) of 10 cm^2. (Assume the force is uniformly distributed over the area. $A = 0.001\ m^2$. Use the formula:
$$\sigma = \frac{F}{A}$$

2. **Civil Engineering**: Determine the bending moment (M) for a simply supported beam with a point load (P) of 2000 N at the midpoint ($L/2$). Assume $L = 8$ m for demonstration purposes. Use the formula:
$$M = \frac{P \times L}{4}$$

3. **Electrical Engineering**: Find the total resistance (R_{total}) of a circuit with three resistors (R1, R2, R3) in series. (Assume the resistors have negligible internal resistance. $R1 = R2 = R3 = 100\,\Omega$). Use the formula:
$$R_{total} = R1 + R2 + R3$$

4. **Chemical Engineering**: Calculate the conversion rate (X) of a chemical reaction with a reactant concentration (C) of 0.5 mol/L after 10 minutes (t). (Assume a first-order reaction and constant reaction conditions. $C_0 = 1.0$ mol/L). Use the formula:
$$X = \frac{C_0 - C_t}{C_0}$$

5. **Aerospace Engineering**: Compute the lift force (L) on an airplane wing with an area (S) of 50 m^2 and a lift coefficient (C_l) of 0.8 at sea level. (Assume steady-state flight and standard atmospheric conditions. $\rho = 1.225\ kg/m^3$). Use the formula:
$$L = \frac{1}{2}\rho V^2 S C_l$$

6. **Environmental Engineering**: Estimate the sedimentation rate (v) of particles in water with a diameter (d) of 0.1 mm and a density (ρ_p) of 2.65 g/cm^3. (Assume laminar flow and spherical particles. $\rho_f = 1000\ kg/m^3$, $\mu = 1\ mPa \cdot s = 10^{-3}\ kg/m \cdot s$.). Use Stokes' Law:
$$v = \frac{2g(\rho_p - \rho_f)d^2}{9\mu}$$

7. **Industrial Engineering**: Determine the cycle time (T) for a production line that produces 100 units per hour (U). (Assume continuous operation and no downtime. $U = 100$ units/hour). Use the formula:
$$T = \frac{60}{U}$$

8. **Materials Engineering**: Calculate the Young's modulus (E) of a material that stretches (ΔL) 0.01 m under a tensile load (F) of 1000 N and has an original length (L_0) of 1 m. (Assume the material behaves linearly and elastically. $A = 100 \times 10^{-6}\ m^2$). Use the formula:
$$E = \frac{FL_0}{A\Delta L}$$

9. **Biomedical Engineering**: Find the voltage (V) across a cell membrane with an internal ion concentration (C_{in}) of 120 mM and an external concentration (C_{out}) of 150 mM. (Assume ideal conditions and negligible membrane permeability to other ions. $R = 8.314\ J/(mol \cdot K)$, $F = 96485\ C/mol$, $T = 298\ K$). Use the Nernst equation:
$$V = \frac{RT}{F} \ln\left(\frac{C_{out}}{C_{in}}\right)$$

10. **Software Engineering**: Calculate the time complexity (T) of a sorting algorithm with an input size (n) and a growth rate of $n \log n$. (Assume average-case scenario and ideal computational resources. $n = 1000$). Use the formula:
$$T(n) = O(n \log n)$$

11. **Thermal Engineering**: Determine the heat transfer rate (Q) through a wall with a thermal conductivity (k) of 0.04 W/mK, an area (A) of 10 m², and a temperature difference (ΔT) of 20°C. (Assume steady-state conditions and uniform material properties. $k = 0.04\,\text{W/mK}$). Use Fourier's Law:
$$Q = kA\Delta T$$

12. **Automotive Engineering**: Compute the braking distance (d) for a car traveling at 60 km/h with a deceleration rate (a) of 5 m/s². (Assume constant deceleration and dry pavement. $v = 60/3.6\,\text{m/s}$). Use the formula:
$$d = \frac{v^2}{2a}$$

13. **Marine Engineering**: Estimate the buoyant force (B) on a ship with a submerged volume (V) of 5000 m³ in seawater. (Assume the ship is fully submerged and in equilibrium. $\rho = 1025\,\text{kg/m}^3$). Use Archimedes' principle:
$$B = \rho V g$$

14. **Nuclear Engineering**: Calculate the half-life ($t_{1/2}$) of an isotope with a decay constant (λ) of 0.693 per year. (Assume constant decay rate and no external influences. $\lambda = 0.693\,\text{per year}$). Use the formula:
$$t_{1/2} = \frac{\ln(2)}{\lambda}$$

15. **Acoustical Engineering**: Find the sound pressure level (L_p) in decibels for a sound wave with an intensity (I) of 1 mW/m². (Assume ideal sound propagation conditions. $I_0 = 10^{-12}\,\text{W/m}^2$). Use the formula:
$$L_p = 10 \log\left(\frac{I}{I_0}\right)$$

16. **Mining Engineering**: Estimate the amount of ore (O) extracted from a mine with a grade (G) of 0.5% and a tonnage (T) of 1000 tons. (Assume homogeneous ore distribution and complete extraction. $G = 0.005$). Use the formula:
$$O = G \times T$$

17. **Petroleum Engineering**: Calculate the flow rate (Q) of oil through a pipeline with a diameter (D) of 0.5 m and a velocity (V) of 1 m/s. (Assume laminar flow and constant viscosity. $\mu = 0.1\,\text{Pa} \cdot \text{s}$). Use the formula:
$$Q = \frac{\pi D^2}{4} V$$

18. **Robotics Engineering**: Find the torque (τ) required for a robotic arm to lift a 5 kg weight at a distance (r) of 0.5 m from the pivot. (Assume no friction or other external forces. $F = 49.05\,\text{N}$). Use the formula:
$$\tau = r \times F$$

19. **Structural Engineering**: Compute the deflection (δ) of a cantilever beam with a length (L) of 2 m and a uniformly distributed load (w) of 300 N/m. (Assume the beam is of uniform cross-section and behaves linearly. $E = 200 \times 10^9\,\text{Pa}, = 6.67 \times 10^{-5}\,\text{m}^4$). Use the formula:
$$\delta = \frac{wL^4}{8EI}$$

20. **Water Resources Engineering**: Calculate the discharge (Q) of a river with a cross-sectional area (A) of 50 m² and an average velocity (V) of 2 m/s. (Assume steady flow and uniform cross-section. $V = 2\,\text{m/s}$). Use the formula:
$$Q = A \times V$$

21. **Renewable Energy Engineering**: Estimate the power output (P) of a solar panel with an efficiency (η) of 15% and an irradiance (E) of 1000 W/m². (Assume ideal conditions and no shading. $A = 10\,\text{m}^2$). Use the formula:
$$P = \eta \times E \times A$$

22. **Control Engineering**: Find the transfer function ($H(s)$) of a control system with a gain (K) of 10 and a time constant (τ) of 5 seconds. (Assume a first-order system and ideal control parameters. $K = 10$). Use the formula:
$$H(s) = \frac{K}{1 + \tau s}$$

23. **Manufacturing Engineering**: Determine the material removal rate (MRR) for a milling process with a feed rate (f) of 0.2 mm/rev and a cutting speed (V) of 100 m/min. (Assume continuous cutting and no tool wear. depth of cut $= 5$ mm). Use the formula:

$$\text{MRR} = f \times V \times \text{depth of cut}$$

24. **Optical Engineering**: Calculate the focal length (f) of a lens with a refractive index (n) of 1.5 and a radius of curvature (R) of 0.2 m. (Assume the lens has a spherical shape and negligible aberrations. $R_1 = R_2 = 0.2\,\text{m}$). Use the lensmaker's equation:

$$\frac{1}{f} = (n-1)\left(\frac{1}{R_1} - \frac{1}{R_2}\right)$$

25. **Telecommunications Engineering**: Find the bandwidth (B) required for a digital signal with a bit rate (R) of 1 Mbps and a signal-to-noise ratio (SNR) of 20 dB. (Assume ideal communication channel and no interference. $R = 1$ Mbps). Use the Shannon-Hartley theorem:

$$B = \frac{R}{\log_2(1+\text{SNR})}$$

26. **Agricultural Engineering**: Estimate the water requirement (WR) for a crop with an evapotranspiration rate (ET) of 5 mm/day and a field area (A) of 1 hectare. (Assume uniform crop characteristics and weather conditions) $A = 10000\,\text{m}^2$). Use the formula:

$$WR = ET \times A$$

27. **Electronics Engineering**: Calculate the gain (A_v) of an operational amplifier (op-amp) circuit with feedback resistors (R_f) and input resistor (R_{in}). (Assume ideal op-amp behavior and no loading effects. $R_{in} = 1\,\text{k}\Omega$, $R_f = 10\,\text{k}\Omega$). Use the formula:

$$A_v = -\frac{R_f}{R_{in}}$$

28. **Chemical Engineering**: Determine the flow rate (Q) of a liquid through a pipe with a diameter (D) and a pressure drop (ΔP). (Assume turbulent flow and constant fluid properties. $D = 0.1$ m, $\Delta P = 100\,\text{kPa}$). Use the Darcy-Weisbach equation:

$$Q = \frac{\pi D^2}{4}\sqrt{\frac{2\Delta P}{\rho}}$$

29. **Structural Engineering**: Compute the natural frequency (f_n) of a vibrating beam with a stiffness (k). (Assume a single degree of freedom and no damping. $m = 2\,\text{kg}$, $k = 5000$ N/m). Use the formula:

$$f_n = \frac{1}{2\pi}\sqrt{\frac{k}{m}}$$

30. **Thermal Engineering**: Estimate the heat transfer rate (Q) from a heated surface with a convective heat transfer coefficient (h) and surface area (A). Assume a temperature difference ($\Delta T = 30\text{K}$). (Assume constant heat transfer coefficient and negligible radiation effects. $h = 100\,\text{W/m}^2\cdot\text{K}$, $A = 0.5\,\text{m}^2$). Use Newton's law of cooling:

$$Q = hA\Delta T$$

31. **Environmental Engineering**: Calculate the dissolved oxygen concentration (C_{DO}) in water based on temperature (T) and atmospheric pressure (P). (Assume Henry's constant kH and ideal gas behavior. $T = 25\,^\circ\text{C}$, $P = 101.3\,\text{kPa}$). Use the Henry's law equation:

$$C_{DO} = kH(T)P$$

32. **Geotechnical Engineering**: Determine the settlement (S) of a foundation on clay soil with a load (P) and a coefficient of consolidation (C_v). (Assume one-dimensional consolidation and isotropic soil behavior. $P = 200\,\text{kN}$, $C_v = 0.02\,\text{m}^2/\text{s}$). Use the formula:

$$S = \frac{P}{C_v}$$

33. **Materials Engineering**: Calculate the fatigue life (N_f) of a component subjected to cyclic loading with stress amplitude (σ_a). (Assume known material properties. $\sigma_f = 400\,\text{MPa}$, $b = 0.1$, $\sigma_a = 100\,\text{MPa}$). Use the S-N curve:

$$N_f = \left(\frac{\sigma_a}{\sigma_f}\right)^{-b}$$

3 | The Logic - Condition & Loop

Logical and loop statements contain expressions

3.1 Logic - The Conditional Control Statement

3.1.1 The if-elif-else Conditional Statement

The Python if statement is typical of if statements in most procedural languages. It takes the form of an if expression, followed by one or more optional elif ("else if") expressions and a final optional else block. The expressions and the else part each have an associated block of nested statements, indented under a header line. When the if statement runs, Python executes the block of code associated with the first expression that evaluates to true, or the else block if all expressions prove false. The general form of an if statement looks like this: **Where, only the block associated with the first true expression in if, elif, and else is executed. To handle multiple true expressions, use separate if statements or combine conditions with logical operators**.

if-elif-else Conditional Control Statement Syntax

```
if expression:     # expression is True
    statements     # Run Associated block
    statements     # Run Associated block
elif expression:   # expression is True
    statements     # Run Associated block
    statements     # Run Associated block
elif expression:   # expression is True
    statements     # Run Associated block
    statements     # Run Associated block
elif expression:   # expression is True
    statements     # Run Associated block
    statements     # Run Associated block
else:              # If No Expression True
    statements     # Run Associated block
    statements     # Run Associated block
```

Example Code

```
a = 200; b = 2000
if b > a:
    c = a + b
    print("b is greater than a")
elif a == b:
    c = a - b
    print("a and b are equal")
elif a == b:
    c = a * b
    print("a and b are equal to")
elif a == b:
    c = a / b
    print("a and b are equal too")
else:
    print("a is greater than b")
```

Example Output

```
b is greater than a
```

3.1.2 try-except-else-finally Statement

Semantically, the block under the try header in this statement represents the main action of the statement—the code you're trying to run and wrap in error processing logic. The except clauses define handlers for exceptions raised during the try block, and the else clause (if coded) provides a handler to be run if no exceptions occur. The var entry here has to do with a feature of raise statements and exception classes, The finally block in a try-except statement ensures that its statements are always executed, regardless of whether an exception occurs or not.

try-except-else-finally Conditional Control Statement Syntax

```
try:
    statements # Run this main action first
except name1:
    statements # Run if name1 is raised in try block
except (name2, name3):
    statements # Run if any of these exceptions occur
except name4 as var:
    statements # Run if name4 is raised, assign to var
except:
    statements # Run for all other exceptions raised
else:
    statements # Run if no exception during try block
finally:
    statements # Always perform this block on exit.
```

Example Code

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Error")
except (IOError,OSError):
    print("IO/OSError")
except Exception as e:
    print("error:", e)
except:
    print("An error")
else:
    print("No Error")
finally:
    print("Finally Run")
```

Example Output

```
Error
Finally Run
```

3.1.3 with-as Context Manager Statement

context_manager_object: This is an object that supports the context management protocol. It is typically responsible for managing some resource, such as opening and closing files or managing database connections. variable: This is an optional variable name to which the result of context_manager_object can be assigned. It may represent the resource managed by the context manager. statements: These are the statements or block of code that you want to execute within the context managed by context_manager_object. These statements will typically operate on the resource managed by the context manager.

with-as Context Manager Statement Syntax

```
with context_manager_object as variable:
    statements  # Run The block
    statements  # Run The block
```

Example Code

```
with open('filename.txt', 'r') as file:
    content = file.read()
    print(content)
```

Example Output

```
Hello, world!
```

A. **raise** : Trigger an exception manually in your code.

B. **assert** : Conditionally trigger an exception in your code.

3.2 Logic - The Loop Control Statement

3.2.1 The for-else Loop Statement

When Python runs a for loop, it assigns the items in the iterable data_value to the target_var one by one and executes the loop body for each. The loop body typically uses the assignment target to refer to the current item in the sequence as though it were a cursor stepping through the sequence. The else part of a for loop in Python is executed if the loop completes normally, meaning it iterates over all items in the iterable data_value without encountering a break statement.

for-else Loop Control Statement Syntax

```
for target_var in data_value: # Assign data items   target_var
    statements                # Repeated loop body: use target
else:                         # Optional else part
    statements                # Run if we didn't hit a 'break'
```

Example Code

```
fruits = [80,90]
for x in fruits:
    print(x)
```

80
90

Advanced Example Code

```
for i, k, l in zip(range(1,3), range(11,13), range(21,23)):
    print(i, k, l)

my_list = ['apple', 'banana', 'orange']
for index, fruit in enumerate(my_list, start=1):
    print(index, fruit)
```

Example Output

```
1 11 21
2 12 22

1 apple
2 banana
3 orange
```

```
# Generator Comprehension Syntax : (expression for item in iterable if condition)
# Set Comprehension Syntax       : {expression for item in iterable if condition}
# List Comprehension Syntax      : [expression for item in iterable if condition]
# Dictionary Comprehension Syntax: {key_expr:val_expr for item in iterable if cond}

square_generator = (x**2 for x in range(5))     # Output: generator object <genexpr>
unique_chars     = {char for char in 'hello'}   # Output: {'e', 'o', 'h', 'l'}
squares          = [x**2 for x in range(5)]     # Output: [0, 1, 4, 9, 16]
square_dict      = {x: x**2 for x in range(5)} # Output: {0:0, 1:1, 2:4, 3:9, 4:16}
```

3.2.2 while-end Loop Statement

The while statement consists of a header line with a test expression, a body of one or more normally indented statements, and an optional else part that is executed if control exits the loop without a break statement being encountered. Python keeps evaluating the test expression at the top and executing the statements nested in the loop body until the test returns a false value.

while-else Loop Control Statement Syntax

```
# Initialize the variable with some initial Data value
while expression: # While expression evaluates to True
    statements    # Execute statements in the loop body
    statements    # Execute statements in the loop body
else:             # Optional else block
    statements    # Run if didn't exit loop with break
```

Example Code

```
i = 1
while i < 6:
    print(i, end=' ')
    i = i + 1
else:
    print("no break")
```

1 2 3 4 5 no break

break : Jumps out of the closest enclosing loop (past the entire loop statement). **continue**: Jumps to the top of the closest enclosing loop (to the loop's header line). **pass** : Does nothing at all: it's an empty statement placeholder. **else** : Runs if and only if the loop is exited normally (without hitting a break).

Write Python scripts to calculate the following problem using conditional statement

1. Write a program to check if a given number is positive, negative, or zero.
2. Create a grading system program where it assigns grades based on percentage marks obtained by students.
3. Write a program to determine if a given year is a leap year or not.
4. Implement a program to find the largest among three numbers entered by the user.
5. Create a program to check if a given character is a vowel or consonant.
6. Write a program to classify a given triangle as equilateral, isosceles, or scalene.
7. Implement a program to check if a given number is prime or composite.
8. Create a program to calculate the roots of a quadratic equation $ax^2 + bx + c = 0$.
9. Write a program to determine if a given number is even or odd.
10. Implement a program to determine the eligibility of a person to vote based on their age.
11. Create a program to calculate the electricity bill for a given number of units consumed.
12. Write a program to check if a given string is a palindrome or not.
13. Implement a program to calculate the area and perimeter of a rectangle or square based on user input.
14. Create a program to determine if a given year is a century year or not.
15. Write a program to determine the largest among four numbers entered by the user.
16. Create a program to calculate the income tax based on the annual income entered by the user.
17. Write a program to check if a given number is divisible by both 2 and 3.
18. Implement a program to determine the eligibility of a person for admission to a university based on their entrance test score.
19. Create a program to classify a given character as uppercase, lowercase, or special symbol.
20. Write a program to calculate the sum of digits of a given number.
21. Implement a program to convert temperature from Celsius to Fahrenheit and vice versa.
22. Create a program to determine the area and circumference of a circle based on the radius entered by the user.
23. Write a program to sort three integers in ascending order.
24. Implement a program to calculate the factorial of a given number using recursion.
25. Create a program to check if a given number is a perfect number or not.
26. Write a program to convert a decimal number to binary, octal, and hexadecimal formats.
27. Implement a program to calculate the sum of all natural numbers up to a given limit.
28. Create a program to generate Fibonacci series up to a specified number of terms.
29. Write a program to check if a given number is an Armstrong number or not.
30. Implement a program to reverse a given number.
31. Create a program to check if a given year is a leap year or not using a different approach.
32. Write a program to determine if a given number is a perfect square or not.
33. Implement a program to calculate the sum of all even numbers and the sum of all odd numbers in a given range.

Write Python scripts to calculate the following series using for loops using the first $n = 20$ terms

Summation of Arithmetic Series: Sum of arithmetic series is to find out the total sum of all the terms in series.

A. Sum of the first n positive integers: $1 + 2 + 3 + \cdots + n$

B. Sum of the first n odd integers: $1 + 3 + 5 + \cdots + (2n - 1)$

C. Sum of the first n even integers: $2 + 4 + 6 + \cdots + 2n$

D. Sum of the squares of the first n positive integers: $1^2 + 2^2 + 3^2 + \cdots + n^2$

E. Sum of the squares of the first n odd integers: $1^2 + 3^2 + 5^2 + \cdots + (2n - 1)^2$

F. Sum of the squares of the first n even integers: $2^2 + 4^2 + 6^2 + \cdots + (2n)^2$

G. Sum of the cubes of the first n positive integers: $1^3 + 2^3 + 3^3 + \cdots + n^3$

Product of Arithmetic Series: Sum of arithmetic series is to find out the total Product of all the terms in series.

A. The product of the first n positive integers is given by: $1 \cdot 2 \cdot 3 \cdot \cdots \cdot n$

B. The product of the first n odd integers is given by: $1 \cdot 3 \cdot 5 \cdot \cdots \cdot (2n - 1)$

C. The product of the first n even integers is given by: $2 \cdot 4 \cdot 6 \cdot \cdots \cdot 2n$

D. Product of squares of the first n positive integers: $\left(1^2\right)\left(2^2\right)\left(3^2\right)\cdots\left(n^2\right)$

E. Product of squares of the first n odd integers: $\left(1^2\right)\left(3^2\right)\left(5^2\right)\cdots\left((2n-1)^2\right)$

F. Product of squares of the first n even integers: $\left(2^2\right)\left(4^2\right)\left(6^2\right)\cdots\left((2n)^2\right)$

G. Product of cubes of the first n positive integers: $\left(1^3\right)\left(2^3\right)\left(3^3\right)\cdots\left(n^3\right)$

H. Product of cubes of the first n odd integers: $\left(1^3\right)\left(3^3\right)\left(5^3\right)\cdots\left((2n-1)^3\right)$

I. Product of cubes of the first n even integers: $\left(2^3\right)\left(4^3\right)\left(6^3\right)\cdots\left((2n)^3\right)$

Series Expressions for π:

A. $1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$

B. $\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots = \frac{\pi^2}{6}$

C. $\frac{1}{1^4} + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \cdots = \frac{\pi^4}{90}$

D. $\frac{1}{1^3} - \frac{1}{3^3} + \frac{1}{5^3} - \frac{1}{7^3} + \cdots = \frac{\pi^3}{32}$

E. $\frac{1}{1^4} + \frac{1}{3^4} + \frac{1}{5^4} + \frac{1}{7^4} + \cdots = \frac{\pi^4}{96}$

F. $\frac{1}{1^5} - \frac{1}{3^5} + \frac{1}{5^5} - \frac{1}{7^5} + \cdots = \frac{5\pi^5}{1536}$

G. $\frac{1}{1^6} + \frac{1}{3^6} + \frac{1}{5^6} + \frac{1}{7^6} + \cdots = \frac{\pi^6}{960}$

H. $\frac{1}{1^7} + \frac{1}{3^7} + \frac{1}{5^7} + \frac{1}{7^7} + \cdots = \frac{\pi^7}{5670}$

I. $\frac{1}{1^8} - \frac{1}{3^8} + \frac{1}{5^8} - \frac{1}{7^8} + \cdots = \frac{7\pi^8}{5760}$

J. $\frac{1}{1^9} + \frac{1}{3^9} + \frac{1}{5^9} + \frac{1}{7^9} + \cdots = \frac{8\pi^9}{945}$

K. $\frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdots = \frac{2}{\pi}$

logarithmic and exponential functions: The Maclaurin series expansions for log and exp functions are:

A. $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$

B. $a^x = 1 + x\ln(a) + \frac{(x\ln(a))^2}{2!} + \frac{(x\ln(a))^3}{3!} + \frac{(x\ln(a))^4}{4!} + \cdots = \sum_{n=0}^{\infty} \frac{(x\ln(a))^n}{n!}$

C. $\log_2(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}(x-1)^n}{n}$

D. $\log_{10}(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}(x-1)^n}{n}$

E. $\ln(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}(x-1)^n}{n}$

Trigonometric Functions Series: The Maclaurin series expansions for some of the basic trigonometric functions:

A. $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!}x^{2n+1}$

B. $\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!}x^{2n}$

C. $\tan(x) = x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \cdots = \sum_{n=1}^{\infty} \frac{B_{2n}(-4)^n(1-4^n)x^{2n-1}}{(2n)!}$, B_{2n} are the Bernoulli numbers.

D. $\csc(x) = \frac{1}{x} + \frac{x}{6} + \frac{7x^3}{360} + \frac{31x^5}{15120} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n(2^{2n-1}-1)B_{2n}x^{2n-1}}{(2n)!}$

E. $\sec(x) = 1 + \frac{x^2}{2} + \frac{5x^4}{24} + \frac{61x^6}{720} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n(2^{2n-1}+1)B_{2n}x^{2n}}{(2n)!}$

F. $\cot(x) = \frac{1}{x} - \frac{x}{3} - \frac{x^3}{45} - \frac{2x^5}{945} - \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}2^{2n}(2^{2n}-1)B_{2n}x^{2n-1}}{(2n)!}$

Inverse Trigonometric Functions: The Maclaurin series expansions for the basic inverse trigonometric functions:

A. $\arcsin(x) = x + \frac{1}{2}\cdot\frac{x^3}{3} + \frac{1\cdot 3}{2\cdot 4}\cdot\frac{x^5}{5} + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\cdot\frac{x^7}{7} + \cdots = \sum_{n=0}^{\infty} \frac{(2n)!}{2^{2n}(n!)^2(2n+1)}x^{2n+1}$

B. $\arccos(x) = \frac{\pi}{2} - \arcsin x = \frac{\pi}{2} - \sum_{n=0}^{\infty} \frac{(2n)!}{2^{2n}(n!)^2(2n+1)}x^{2n+1}$

C. $\arctan(x) = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}x^{2n+1}$

D. $\operatorname{arccot}(x) = \frac{\pi}{2} - \arctan x = \frac{\pi}{2} - \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}x^{2n+1}$

E. $\operatorname{arcsec}(x) = \arccos\frac{1}{x} = \sum_{n=0}^{\infty} \frac{(2n)!}{2^{2n}(n!)^2(2n+1)}\frac{1}{x^{2n+1}}$

F. $\operatorname{arccsc}(x) = \arcsin\frac{1}{x} = \sum_{n=0}^{\infty} \frac{(2n)!}{2^{2n}(n!)^2(2n+1)}\frac{1}{x^{2n+1}}$

Hyperbolic Functions: The Maclaurin series expansions for some of the basic hyperbolic functions:

A. $\sinh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \cdots = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!}$

B. $\cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}$

C. $\tanh(x) = x - \frac{x^3}{3} + \frac{2x^5}{15} - \frac{17x^7}{315} + \cdots = \sum_{n=0}^{\infty} \frac{B_{2n}(-4)^n(1-4^n)}{(2n)!} x^{2n-1}$, B_{2n} are the Bernoulli numbers.

D. $\coth(x) = \frac{1}{x} + \frac{x}{3} + \frac{x^3}{45} + \frac{2x^5}{945} + \cdots = \sum_{n=0}^{\infty} \frac{2^{2n}|B_{2n}|(2^{2n}-1)}{(2n)!} x^{2n-1}$

E. $\operatorname{sech}(x) = 1 - \frac{x^2}{2!} + \frac{5x^4}{4!} - \frac{61x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n(2n)!}{(2^n n!)^2} \frac{x^{2n}}{(2n)!}$

F. $\operatorname{csch}(x) = \frac{1}{x} - \frac{x}{6} - \frac{x^3}{40} - \frac{x^5}{3456} - \cdots = \sum_{n=0}^{\infty} (-1)^n \frac{(2n)!}{(2^n n!)^2} \frac{x^{2n-1}}{(2n-1)!}$

Inverse Hyperbolic Functions: The Maclaurin series expansions for all of the inverse hyperbolic functions:

A. $\operatorname{arsinh}(x) = x + \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6 \cdot 7} x^7 + \cdots = \sum_{n=0}^{\infty} \frac{(2n-1)!!}{2^n n!} x^{2n+1}$

B. $\operatorname{arcosh}(x) = \ln(x + \sqrt{x^2-1}) = x - \frac{1}{2 \cdot 2} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 3} x^5 - \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6 \cdot 5} x^7 + \cdots = \sum_{n=0}^{\infty} \frac{(2n)!}{2^n (n!)^2} \frac{x^{2n}}{(2n-1)!!}$

C. $\operatorname{artanh}(x) = \frac{1}{2} \ln\left(\frac{1+x}{1-x}\right) = x + \frac{1}{3} x^3 + \frac{1}{5} x^5 + \frac{1}{7} x^7 + \cdots = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1}$

D. $\operatorname{arcoth}(x) = \frac{1}{2} \ln\left(\frac{x+1}{x-1}\right) = \frac{1}{x} + \frac{1}{3} x + \frac{1}{5} x^3 + \frac{1}{7} x^5 + \cdots = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1}$

E. $\operatorname{arsech}(x) = \ln\left(\frac{1+\sqrt{1-x^2}}{x}\right) = \frac{\pi}{2} - x - \frac{1}{2 \cdot 2} x^3 - \frac{1 \cdot 3}{2 \cdot 4 \cdot 3} x^5 - \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6 \cdot 5} x^7 - \cdots = \sum_{n=0}^{\infty} \frac{(2n)!}{2^n (n!)^2} \frac{x^{2n}}{(2n+1)!!}$

F. $\operatorname{arcsch}(x) = \ln\left(\frac{1}{x} + \sqrt{\frac{1}{x^2}+1}\right) = x + \frac{1}{6} x^3 + \frac{3}{40} x^5 + \frac{5}{112} x^7 + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{(n!)^2 (2n+1)} x^{2n+1}$

Here is the general approach to writing Python scripts for calculating each of the series expressions:

I. Identify the pattern: Understand the mathematical pattern or formula behind each series expression.

II. Generate the Terms: Set up a for loop to iterate through the terms (Repeated Unit) of the series.

III. Accumulate the sum or product: Within the loop, accumulate the sum or product of those series.

$\text{sum}(n) \equiv 1 + 2 + 3 + \ldots + n$. calculate the value of summation up to 6

```
x        = 6                    # To find the value of 6! (factorial(6))
multi_01 = 0                    # Assume a reasonable predefined value

for n in range(1,x+1):          # Iterate through the terms of the series
    print(n)                    # Calculate the value of each term
    multi_01 = multi_01 + n     # Accumulate the sum of the terms
```

$n! \equiv 1 \times 2 \times 3 \times \ldots \times n$. calculate the value of 6! (factorial(6))

```
x        = 6                    # To find the value of 6! (factorial(6))
multi_01 = 1                    # Assume a reasonable predefined value

for n in range(1,x+1):          # Iterate through the terms of the series
    print(n)                    # Calculate the value of each term
    multi_01 = multi_01 * n     # Accumulate the product of the terms
```

4 | The Function - Input & Output

Functions are reusable statements that take input and produce output

4.1 Function - The Definition and Types

Definition: In mathematics, Function is an expression, rule, or law that defines a relationship between one variable (the independent one) to another variable (the dependent one). Functions are ubiquitous in mathematics and are essential for formulating physical relationships in the sciences. **Notation:** A function is represented as $y = f(x)$. The function (f) relates elements in one set (independent set/variable, x) to elements in another set (dependent set/variable y) through a rule called f. The set of input values x is called the domain, and the set of corresponding output values y or $f(x)$ is called the range, where y or $f(x)$ and x are related such that for each x, there is a unique value of y or $f(x)$.

Input - Independent set/variable, x	Process - Function, $f \equiv x^2$	Output - Dependent set/variable, y or $f(x)$
1	Machine	1
2		4
3		9

Types of Function

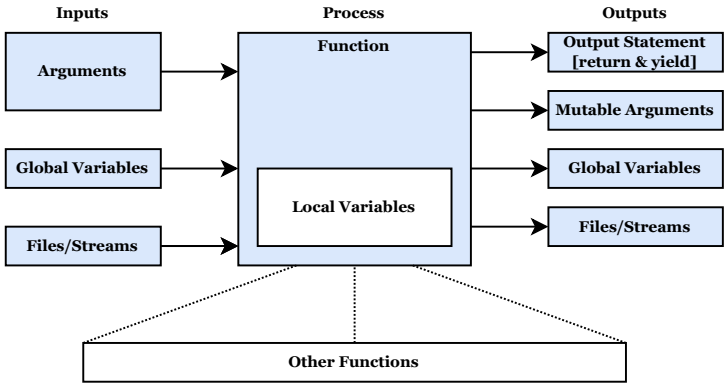
A. Algebraic or Polynomial Function
   a. Single Variable Function
      i. Polynomial Function: Constant, Linear, Quadratic ….
      ii. Radical Function
      iii. Rational Function
   b. Bi-variate Function
   c. Multivariate Function

B. Transcendental Function
   a. Exponential Function - (..., e)
   b. Logarithmic Function - (log, ln)
   c. Trigonometric Function - (sin, cosec, cos, sec, tan, cot)
   d. Hyperbolic Function - (sinh, cosech, cosh, sech, tanh, coth)

C. Changing and Accumulating Function
   a. Differentiating Function - $\frac{d}{dx}\cdots$
   b. Integrating Function - $\int \cdots dx$

4.1.1 Function - The Definition and Types

In Programming, A function is a block of code which only runs when it is called. One can pass data, known as parameters/independent variables/arguments, into a function. A function can return data as a result. In simple terms, a function is a process that groups a set of statements so they can be run more than once in a program—a packaged procedure invoked by name. There are two types of functions in Python: Firstly, **Built-in Functions** - These are pre-defined functions in Python. Secondly, **User-Defined Functions** - these types of functions are defined by the user to perform any specific task.



4.1.2 Function - The Why We Need

A. **For Single Function:** Maximizing code reuse and minimizing redundancy

B. **For Multiple Function:** Procedural decomposition

4.1.3 Function - The Creation and Calling

The Python Based Builtin Function (Created)

```
'abs', 'aiter',  'all',  'anext', 'any', 'ascii', 'bin',
'bool', 'breakpoint',  'bytearray', 'bytes', 'callable',
'chr', 'classmethod', 'compile', 'complex', 'copyright',
'credits', 'delattr','dict', 'dir', 'display', 'divmod',
'enumerate', 'eval','exec', 'execfile','filter','float',
'format', 'frozenset','get_ipython','getattr','globals',
'hasattr', 'hash', 'help',  'hex', 'id', 'input', 'int',
'isinstance',  'issubclass',  'iter',  'len', 'license',
'list', 'locals', 'map','max','memoryview','min','next',
'object', 'oct', 'open','ord', 'pow','print','property',
'range', 'repr', 'reversed',  'round', 'runfile', 'set',
'setattr', 'slice', 'sorted','staticmethod','str','sum',
'super', 'tuple', 'type', 'vars', 'zip'.
```

Example Code (Call)

```
print(len("Hello"))
var1 = sorted([3,4,1])
for i in range(5):
    print(i)

name = input("Type: ")
print("Hello,", name)

l = [1, 2, 3];
v=map(lambda x:x**2,l)

print(abs(-5))
x = 42; print(id(x))
```

Single Line User Defined Function

```
fun_name = lambda arg1, ..., argn: expression    # Create
var_name = fun_name(arg1, ..., argn)             # Call
```

Example Code

```
x=lambda a,b,c:a*b**c
var_01 = x(1,2,4)
```

Multiple Line User Defined Function

```
def fun_name(arg1, ..., argn):                   # Create
    statements;
    return value1, ..., valuen                   # Opt[2]
var1, ..., varn = fun_name(argval1, ..., argvaln)# Call
```

Example Code

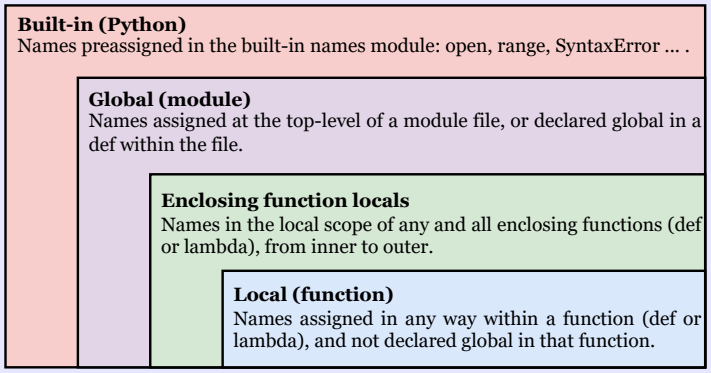
```
def my_add(a,b):
    c  = a+b
    return a,b,c
[x,_,y] = my_add(2,3)
```

4.2 Function - The Creation and Calling

4.2.1 The Function Syntax - Argument Scopes and Matching

Argument Scopes - The king is powerful only in his own territory.

LEGB Arguments scope lookup rule



Example Code

```
x = 0              # Global
def f1():          # Enclosed
    x = 1
    print("f1: x =", x)
    def f2():    # Local
        global y; y = 3
        x = 2
        print("f2: x =",x)
    f2()
f1()
print("O1: x =", x)
print("I1: y =", y)
```

In Python, when a variable is referenced, searches for it in a specific order known as the LEGB scope lookup rule: **Local scope**: Python first searches for the variable in the local scope, which is the innermost scope, typically within a function. **Enclosing function's local scopes**: If the variable is not found in the local scope, Python searches in any enclosing functions' local scopes. This applies to nested functions. **Global scope**: If the variable is still not found, Python looks in the global scope, which is the outermost scope of the module or script. **Built-in scope**: Finally, if the variable is not found in any of the above scopes, Python searches in the built-in scope, which contains built-in functions and objects.

Argument Matching - Our territory, our rule

Functions can have zero, one, or more arguments. Arguments are optional. There are two types of arguments - :

A. **Constant length arguments:** Required or Positional Arguments, Default arguments, Keyword arguments

B. **Variable-length args:** Arbitrary Positional Arguments (*args), Arbitrary Keyword Arguments (**kwargs)

Syntax	Location	Interpretation
def func(name)	Function	Normal argument: matches any passed value by position or name
def func(name=value)	Function	Default argument value, if not passed in the call
def func(*name)	Function	Matches and collects remaining positional arguments in a tuple
def func(**name)	Function	Matches and collects remaining keyword arguments in a dictionary
def func(*other, name)	Function	Arguments that must be passed by keyword only in calls (3.X)
def func(*, name=value)	Function	Arguments that must be passed by keyword only in calls (3.X)

```
def fun_name(arg1, arg2, *args, **kargws) # Fun def                    # Create
    ..................................    # Fun Body  - Processing
    ..................................    # Fun Body  - return, yield
fun_name(val1, val2, val3, ..., valn, var1 = valx, ..., varn = varz)   # Calling
```

4.2.2 The Function Example

```
def fun10(a, b, c=0, *u, **v): # Create
    print("a (normal  argument):", a); print("b (normal  argument):", b)
    print("c (default argument):", c); print("u (tuple   argument):", u)
    print("v (dictionary  args):", v)
fun10(1, 2, 3, 4, 5, x=6, y=7) # Calling
```

A. **Circle Area Problem:** Write a function to calculate the area of a circle given the radius (r) for the equation, $A = \pi r^2$. then call the function as example: `calculate_circle_area(5)`

B. **Sphere Volume Problem:** Create a function to compute the volume of a sphere using its radius (r) for the equation, $V = \frac{4}{3}\pi r^3$. then call the function as example: `calculate_sphere_volume(3)`

C. **Cylinder Surface Area Problem:** Develop a function that returns the surface area of a cylinder given its radius (r) and height (h) for the equation, $A = 2\pi r(r + h)$. then call the function as example: `calculate_cylinder_surface_area(4, 10)`

D. **Cylinder Volume Problem:** Construct a function to find the volume of a cylinder using the radius (r) and height (h) for the equation, $V = \pi r^2 h$. then call the function as example: `calculate_cylinder_volume(4, 10)`

E. **Rectangle Area Problem:** Design a function that determines the area of a rectangle with length (l) and width (w) for the equation, $A = l \times w$. then call the function as example: `calculate_rectangle_area(7, 5)`

F. **Rectangular Prism Volume Problem:** Formulate a function that calculates the volume of a rectangular prism using its length (l), width (w), and height (h) for the equation, $V = l \times w \times h$. then call the function as example: `calculate_rectangular_prism_volume(7, 5, 3)`

G. **Triangle Area Problem:** Implement a function to calculate the area of a triangle given its base (b) and height (h) for the equation, $A = \frac{1}{2} b \times h$. then call the function as example: `calculate_triangle_area(8, 6)`

H. **Cone Volume Problem:** Set up a function to determine the volume of a cone with a base radius (r) and height (h) for the equation, $V = \frac{1}{3}\pi r^2 h$. then call the function as example: `calculate_cone_volume(3, 5)`

I. **Sphere Surface Area Problem:** Establish a function that calculates the surface area of a sphere based on its radius (r) for the equation, $A = 4\pi r^2$. then call the function as example: `calculate_sphere_surface_area(3)`

J. **Pyramid Volume Problem:** Generate a function that finds the volume of a pyramid with a base length (l), base width (w), and height (h) for the equation, $V = \frac{1}{3} l \times w \times h$. then call the function as example: `calculate_pyramid_volume(4, 6, 5)`

Write Python function that will calculate the following series using for loops using first $n = 20$ terms

logarithmic and exponential functions: The Maclaurin series expansions for log and exp functions are:

A. $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$

B. $a^x = 1 + x\ln(a) + \frac{(x\ln(a))^2}{2!} + \frac{(x\ln(a))^3}{3!} + \frac{(x\ln(a))^4}{4!} + \cdots = \sum_{n=0}^{\infty} \frac{(x\ln(a))^n}{n!}$

C. $\log_2(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}(x-1)^n}{n}$

D. $\log_{10}(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}(x-1)^n}{n}$

E. $\ln(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}(x-1)^n}{n}$

Trigonometric Functions Series: The Maclaurin series expansions for some of the basic trigonometric functions:

A. $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$

B. $\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$

C. $\tan(x) = x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \cdots = \sum_{n=1}^{\infty} \frac{B_{2n}(-4)^n(1-4^n)x^{2n-1}}{(2n)!}$, B_{2n} are the Bernoulli numbers.

D. $\csc(x) = \frac{1}{x} + \frac{x}{6} + \frac{7x^3}{360} + \frac{31x^5}{15120} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n(2^{2n-1}-1)B_{2n}x^{2n-1}}{(2n)!}$

E. $\sec(x) = 1 + \frac{x^2}{2} + \frac{5x^4}{24} + \frac{61x^6}{720} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n(2^{2n-1}+1)B_{2n}x^{2n}}{(2n)!}$

F. $\cot(x) = \frac{1}{x} - \frac{x}{3} - \frac{x^3}{45} - \frac{2x^5}{945} - \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}2^{2n}(2^{2n}-1)B_{2n}x^{2n-1}}{(2n)!}$

Inverse Trigonometric Functions: The Maclaurin series expansions for the basic inverse trigonometric functions:

A. $\arcsin(x) = x + \frac{1}{2}\cdot\frac{x^3}{3} + \frac{1\cdot 3}{2\cdot 4}\cdot\frac{x^5}{5} + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\cdot\frac{x^7}{7} + \cdots = \sum_{n=0}^{\infty} \frac{(2n)!}{2^{2n}(n!)^2(2n+1)}x^{2n+1}$

B. $\arccos(x) = \frac{\pi}{2} - \arcsin x = \frac{\pi}{2} - \sum_{n=0}^{\infty} \frac{(2n)!}{2^{2n}(n!)^2(2n+1)}x^{2n+1}$

C. $\arctan(x) = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}x^{2n+1}$

D. $\operatorname{arccot}(x) = \frac{\pi}{2} - \arctan x = \frac{\pi}{2} - \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}x^{2n+1}$

E. $\operatorname{arcsec}(x) = \arccos\frac{1}{x} = \sum_{n=0}^{\infty} \frac{(2n)!}{2^{2n}(n!)^2(2n+1)}\frac{1}{x^{2n+1}}$

F. $\operatorname{arccsc}(x) = \arcsin\frac{1}{x} = \sum_{n=0}^{\infty} \frac{(2n)!}{2^{2n}(n!)^2(2n+1)}\frac{1}{x^{2n+1}}$

Hyperbolic Functions: The Maclaurin series expansions for some of the basic hyperbolic functions:

A. $\sinh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \cdots = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!}$

B. $\cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}$

C. $\tanh(x) = x - \frac{x^3}{3} + \frac{2x^5}{15} - \frac{17x^7}{315} + \cdots = \sum_{n=0}^{\infty} \frac{B_{2n}(-4)^n(1-4^n)}{(2n)!}x^{2n-1}$, B_{2n} are the Bernoulli numbers.

D. $\coth(x) = \frac{1}{x} + \frac{x}{3} + \frac{x^3}{45} + \frac{2x^5}{945} + \cdots = \sum_{n=0}^{\infty} \frac{2^{2n}|B_{2n}|(2^{2n}-1)}{(2n)!}x^{2n-1}$

E. $\operatorname{sech}(x) = 1 - \frac{x^2}{2!} + \frac{5x^4}{4!} - \frac{61x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n(2n)!}{(2^n n!)^2}\frac{x^{2n}}{(2n)!}$

F. $\operatorname{csch}(x) = \frac{1}{x} - \frac{x}{6} - \frac{x^3}{40} - \frac{x^5}{3456} - \cdots = \sum_{n=0}^{\infty}(-1)^n\frac{(2n)!}{(2^n n!)^2}\frac{x^{2n-1}}{(2n-1)!}$

Inverse Hyperbolic Functions: The Maclaurin series expansions for all of the inverse hyperbolic functions:

A. $\operatorname{arsinh}(x) = x + \frac{1}{2\cdot 3}x^3 + \frac{1\cdot 3}{2\cdot 4\cdot 5}x^5 + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6\cdot 7}x^7 + \cdots = \sum_{n=0}^{\infty} \frac{(2n-1)!!}{2^n n!}x^{2n+1}$

B. $\operatorname{arcosh}(x) = \ln(x+\sqrt{x^2-1}) = x - \frac{1}{2\cdot 2}x^3 + \frac{1\cdot 3}{2\cdot 4\cdot 3}x^5 - \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6\cdot 5}x^7 + \cdots = \sum_{n=0}^{\infty} \frac{(2n)!}{2^n(n!)^2}\frac{x^{2n}}{(2n-1)!!}$

C. $\operatorname{artanh}(x) = \frac{1}{2}\ln\left(\frac{1+x}{1-x}\right) = x + \frac{1}{3}x^3 + \frac{1}{5}x^5 + \frac{1}{7}x^7 + \cdots = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1}$

D. $\operatorname{arcoth}(x) = \frac{1}{2}\ln\left(\frac{x+1}{x-1}\right) = \frac{1}{x} + \frac{1}{3}x + \frac{1}{5}x^3 + \frac{1}{7}x^5 + \cdots = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1}$

E. $\operatorname{arsech}(x) = \ln\left(\frac{1+\sqrt{1-x^2}}{x}\right) = \frac{\pi}{2} - x - \frac{1}{2\cdot 2}x^3 - \frac{1\cdot 3}{2\cdot 4\cdot 3}x^5 - \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6\cdot 5}x^7 - \cdots = \sum_{n=0}^{\infty} \frac{(2n)!}{2^n(n!)^2}\frac{x^{2n}}{(2n+1)!!}$

F. $\operatorname{arcsch}(x) = \ln\left(\frac{1}{x} + \sqrt{\frac{1}{x^2}+1}\right) = x + \frac{1}{6}x^3 + \frac{3}{40}x^5 + \frac{5}{112}x^7 + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n(2n)!}{(n!)^2(2n+1)}x^{2n+1}$

5 | The Box - OOPs & Groups

Modules contain all the statements

Builtins Functions: '__IPYTHON__', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'aiter', 'all', 'anext', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', **'classmethod'**, 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate', 'eval', 'exec', 'execfile', 'filter', 'float', 'format', 'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', **'property'**, 'range', 'repr', 'reversed', 'round', 'runfile', 'set', 'setattr', 'slice', 'sorted', **'staticmethod'**, 'str', 'sum', **'super'**, 'tuple', 'type', 'vars', 'zip',

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

5.1 The OOPs

5.1.1 OOPs - The What

Object Oriented Programmings (OOPs) - The Definition ≡ Data (Attributes) + Functions (Methods)

Object-oriented programming (OOP) is a programming paradigm based on the concept of objects, Which can contain data and code: data in the form of **fields** (often known as **independent variables** or **attributes** or **properties** or **data**), and code in the form of **procedures** (often known as **functions** or **methods** or **Behaviours** or **actions**). In short, a class is a blueprint, template, or prototype that defines how an object will behave.

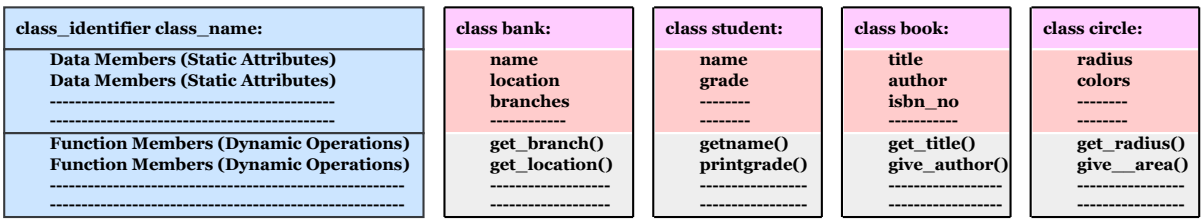


Fig. 5.1: One can conceptualize a class as a "3-compartment box" where the class name, data attributes, and methods are encapsulated together, providing a blueprint for creating objects with consistent data and functions.

5.1.2 OOPs - The Why

A. **Encapsulation:** Encapsulation is the Grouping of Information (Attributes and Methods)

B. **Polymorphism:** Polymorphism is the Redefining of Information (Attributes and Methods)

C. **Inheritance:** Inheritance is the Sharing of Information (Attributes and Methods)

D. **Abstraction:** Abstraction is the Hiding of Information (Attributes and Methods)

5.1.3 OOPs - The How

Attributes/Variables Inside a class:

A. **Class Attributes/Variables:** A class variable is a variable that is declared inside of a class but outside of any instance method or `__init__()` method.

B. **Instance or Object Attributes/Variables:** The instance variables are attributes attached to an instance of a class. We define instance variables in the constructor (`__init__()` method of a class).

Methods Inside a Class: Using decorator

A. **Instance Method:** Used to access or modify the object attributes. If we use instance variables inside a method, such methods are called instance methods.

B. **Class Method:** Used to access or modify the class state. In method implementation, if we use only class variables, then such type of methods we should declare as a class method.

C. **Static Method:** It is a general utility method that performs a task in isolation. Inside this method, we don't use instance or class variables because this static method doesn't have access to the class attributes.

```
class my_class:
    attr_01   = 10                              # Public class  attribute
    _attr_02  = 20                              # Protected class   attr.
    __attr_03 = 30                              # Private class attribute
    def __init__(self, attr_04, attr_05, attr_06):
        self.attr_04   = attr_04                # Public instance  attribute
        self._attr_05  = attr_05                # Protected instance   attr.
        self.__attr_06 = attr_06                # Private instance attribute
    def method_01(self):                        # Public    method
        return self.attr_01 + self.attr_04      # ----------------
    def _method_02(self):                       # Protected method
        return self._attr_02 - self._attr_05    # ----------------
    def __method_03(self):                      # Private   method
        return self.__attr_03 * self.__attr_06 # ----------------
    @property                                   # property method
    def method_04(self):                        # ---------------
        return self.attr_01 + 2                 # ---------------
    @classmethod                                # Class    method
    def method_05(cls):                         # ---------------
        return cls._attr_02 - 2                 # ---------------
    @staticmethod                               # Static   method
    def method_06(parameter):                   # ---------------
        return parameter * 2                    # ---------------

obj = my_class(5, 8, 10) # Creating an object of the my_class class  # -----------
print("method_01:", obj.method_01())                                 # Output:  15
print("method_02:", obj._method_02())                                # Output:  12
print("method_03:", obj._my_class__method_03())                      # Output: 300
print("method_04 (property method):", obj.method_04)                 # Output:  12
print("method_05 (class method):", my_class.method_05())             # Output:  18
print("method_06 (static method):", my_class.method_06(5))           # Output:  10
```

Encapsulation - The Grouping of Information

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP), including abstraction, inheritance, and polymorphism. This lesson will cover what encapsulation is and how to implement it in Python. Encapsulation can be achieved by declaring the data members and methods of a class either as private or protected. But In Python, we don't have direct access modifiers like public, private, and protected. We can achieve this by using single underscore and double underscores. Access modifiers limit access to the variables and methods of a class. Python provides three types of access modifiers public, protected, and private.

A. **Public:** Accessible anywhere from outside of class. Example: attr, method

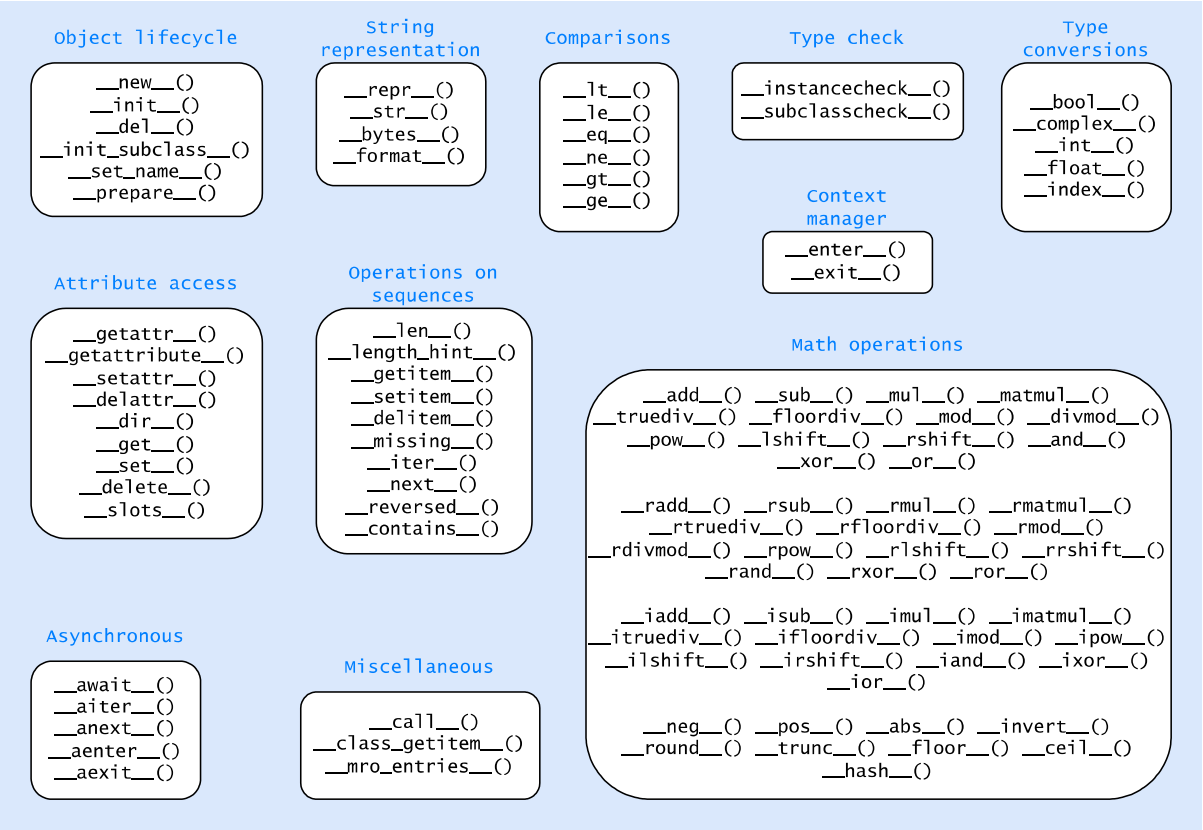
B. **Protected:** Accessible within the class and its sub-classes. _attr, _method

C. **Private:** Accessible within the class. Example: __attr, __method

```
def uppercase_name(func):                          # Define a decorator function
    def wrapper(self, name):
        uppercase_name = name.upper()
        return func(self, uppercase_name)
    return wrapper

class Employee:                                    # Define a class called Employee
    def __init__(self, name, project, salary): # Define the constructor method
        self.name     = name                       # Public member
        self._project = project                    # Protected member
        self.__salary = salary                     # Private member

    @property                                      # Define a property for salary
    def salary(self):                              # Getter for private member
        return self.__salary

    @salary.setter                                 # Define a setter for salary
    def salary(self, new_salary):                  # Setter for private member
        self.__salary = new_salary

    @staticmethod                                  # Define a static method
    def is_valid_salary(salary):                   # Static method to check
        return salary >= 0

    @classmethod                                   # Define a class method
    def from_string(cls, emp_str):                 # Class method to create
        name, project, salary = emp_str.split(",")
        return cls(name, project, int(salary))

    def display_info(self):
        print(f"Name: {self.name},Project: {self._project},Salary: {self.salary}")

    @uppercase_name
    def set_name(self, name):                      # Define a method to set the name
        self.name = name

emp = Employee("Alice", "ProjectX", 50000)         # Create an instance
print("Name:", emp.name)                           # Print the employee's name
emp.set_name("Bob")                                # Set the employee's name
print("Updated Name:", emp.name)                   # Print the updated name


emp2 = Employee.from_string("MR,ProjectY,70") # Create instance form str
emp2.display_info()                           # Display the informations
```

Polymorphism - The Redefining of Information

Polymorphism in Python refers to the ability of objects to exhibit different behaviors or have different implementations of methods depending on their specific type or class. Example: 2+2 = 4 but “Steven” + “Brunton” = “StevenBrunton”.




```
class Fraction:
    def __init__(self, num, den):
        self.num = num
        self.den = den
    def __str__(self):
        return f"{self.num}/{self.den}"
    def __add__(self, other):
        new_num = self.num * other.den + other.num * self.den
        new_den = self.den * other.den
        return Fraction(new_num, new_den)
    def __sub__(self, other):
        new_num = self.num * other.den - other.num * self.den
        new_den = self.den * other.den
        return Fraction(new_num, new_den)
    def __mul__(self, other):
        new_num = self.num * other.num
        new_den = self.den * other.den
        return Fraction(new_num, new_den)
    def __truediv__(self, other):
        new_num = self.num * other.den
        new_den = self.den * other.num
        return Fraction(new_num, new_den)
fraction_01 = Fraction(3, 4); fraction_02 = Fraction(1, 2)
print("Fraction 1    :", fraction_01,"Fraction 2    :", fraction_02) # Output: 3/4
print("Addition      :", fraction_01 + fraction_02)                   # Output: 5/4
print("Subtraction   :", fraction_01 - fraction_02)                   # Output: 1/4
print("Multiplication:", fraction_01 * fraction_02)                   # Output: 3/8
print("Division      :", fraction_01 / fraction_02)                   # Output: 3/2
```

Inheritance - The Sharing of Information

The process of inheriting the properties of the **super class (parent class)** into a **child class** is called inheritance. The existing class is called a base class or parent class and the new class is called a subclass or child class or derived class. Different types of inheritances are

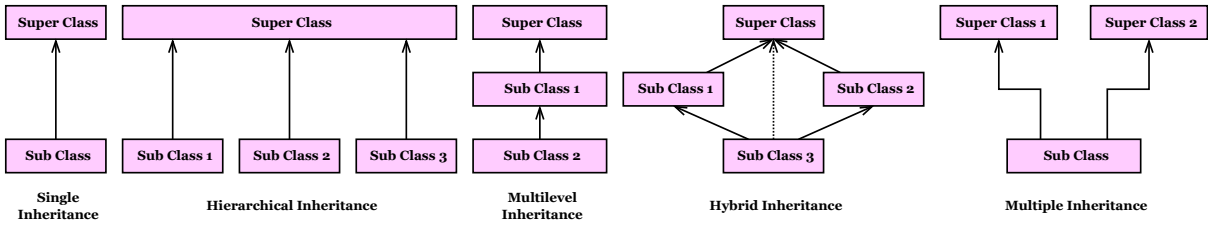
A. **Single inheritance:** A derived class inherits from one base class.

B. **Multiple inheritance:** A derived class inherits from multiple base classes.

C. **Multilevel inheritance:** A derived class inherits from a base class that inherits from another base class.

D. **Hierarchical inheritance:** Multiple derived classes inherit from the same base class.

E. **Hybrid inheritance:** A combination of two or more of the above inheritance types.




```
class Vehicle:
    def vehicle_info(self):      # Base class method
        print("Inside Vehicle class")
class Car(Vehicle):
    def car_info(self):          # Derived class method
        print("Inside Car class")
class Truck(Vehicle):
    def truck_info(self):        # Derived class method
        print("Inside Truck class")
class SportsCar(Car, Vehicle):   # Multiple Inheritance
    def sports_car_info(self):   # Derived class method
        print("Inside SportsCar class")
class SuperCar(SportsCar):       # Multilevel Inheritance
    def super_car_info(self):    # Derived class method
        print("Inside SuperCar class")
class RacingCar(Car):             # Hierarchical Inheritance
    def racing_car_info(self):   # Derived class method
        print("Inside RacingCar class")

# Instances of each class created and methods called ----------------------------
s_car = SportsCar()          # Instance of SportsCar class
s_car.vehicle_info()         # Accessing base class method (Single Inheritance)
s_car.car_info()             # Accessing derived class method (Single Inheritance)
s_car.sports_car_info()      # Accessing derived class method (Hybrid Inheritance)

s_upercar = SuperCar()       # Instance of SuperCar class
s_upercar.vehicle_info()     # Accessing base class method (Single Inheritance)
s_upercar.car_info()         # Accessing derived class method (Single Inheritance)
s_upercar.sports_car_info() # Accessing derived class method (Hybrid Inheritance)
s_upercar.super_car_info()  # Accessing derived class method (Multilevel Inher. )

r_car = RacingCar()          # Instance of RacingCar class
r_car.vehicle_info()         # Accessing base class method (Single Inheritance)
r_car.car_info()             # Accessing derived class method (Single Inheritance)
r_car.racing_car_info()      # Accessing derived class method (Hierarchical Inher)
```

Abstraction - The Hiding of Information

Abstraction focuses on hiding the internal implementations of a process or method from the user. In this way, the user knows what he is doing but not how the work is being done. Three examples of abstraction are given here:

```
from abc import ABC, abstractmethod

class AbstractClass(ABC):
    @abstractmethod
    def abstract_method(self):
        pass
class ConcreteClass(AbstractClass):
    def abstract_method(self):
        # Implementation of the abstract method
        pass
```

```
from abc import ABC, abstractmethod

class Shape(ABC):                        # Abstract class defining the interface
    @abstractmethod
    def area(self):
        pass
class Circle(Shape):                     # Concrete subclass implementing the Shape
    def __init__(self, radius):
        self.radius = radius
    def area(self):                      # Implementation of the abstract method
        return 3.14 * self.radius * self.radius

circle = Circle(5)                       # Create an instance of the concrete subclass
print("Circle Area:", circle.area())  # Output: Area of circle: 78.5
```

```
import math; from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
    @abstractmethod
    def perimeter(self):
        pass
class Rectangle(Shape):  # A class representing a rectangle
    def __init__(self, width, height):
        self.width  = width
        self.height = height
    def area(self):      # Calculate the area of the rectangle
        return self.width * self.height
    def perimeter(self): # Calculate the perimeter of the rectangle
        return 2 * (self.width + self.height)
class Circle(Shape):     # A class representing a circle
    def __init__(self, radius):
        self.radius = radius
    def area(self):      # Calculate the area of the circle
        return math.pi * self.radius ** 2
    def perimeter(self): # Calculate the circumference of the circle
        return 2 * math.pi * self.radius

rectangle = Rectangle(5, 4); circle = Circle(3)       # Create instances
print(rectangle.area(), rectangle.perimeter())        # Output:  20, 18
print(circle.area(), circle.perimeter())              # Output:  28.274.., 18.849..
```

5.2 The Groups

5.2.1 Group - The Modules and Packages

Python Modules: A module is basically a bunch of related code saved in a file with the extension .py. You may choose to define functions, classes, or variables in a module. It's also fine to include runnable code in modules. A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended. Within a module, the module's name (as a string) is available as the value of the global variable __name__. For instance, use your favorite text editor to create a file called mymath.py in the current directory with the following contents:

mymath.py

```
def addition(x, y):
    """Returns the sum of two numbers."""
    return x + y

def subtraction(x, y):
    """Returns the difference between two numbers."""
    return x - y

def multiplication(x, y):
    """Returns the product of two numbers."""
    return x * y

def division(x, y):
    """Returns the result of dividing one number by another."""
    if y == 0:
        raise ValueError("Division by zero is not allowed.")
    return x / y
```

Example Code

```
import mymath     # Import the mymath module

print("Addition:", mymath.addition(5, 3))

print("Subtraction:", mymath.subtraction(10, 4))

print("Multiplication:", mymath.multiplication(2, 6))

print("Division:", mymath.division(15, 3))
```

A module in Python is a single file that contains Python code in the form of functions, executable statements, variables, and classes. A module acts as a self-contained unit of code that can be imported and used in other programs or modules. Creating a module in Python is straightforward:

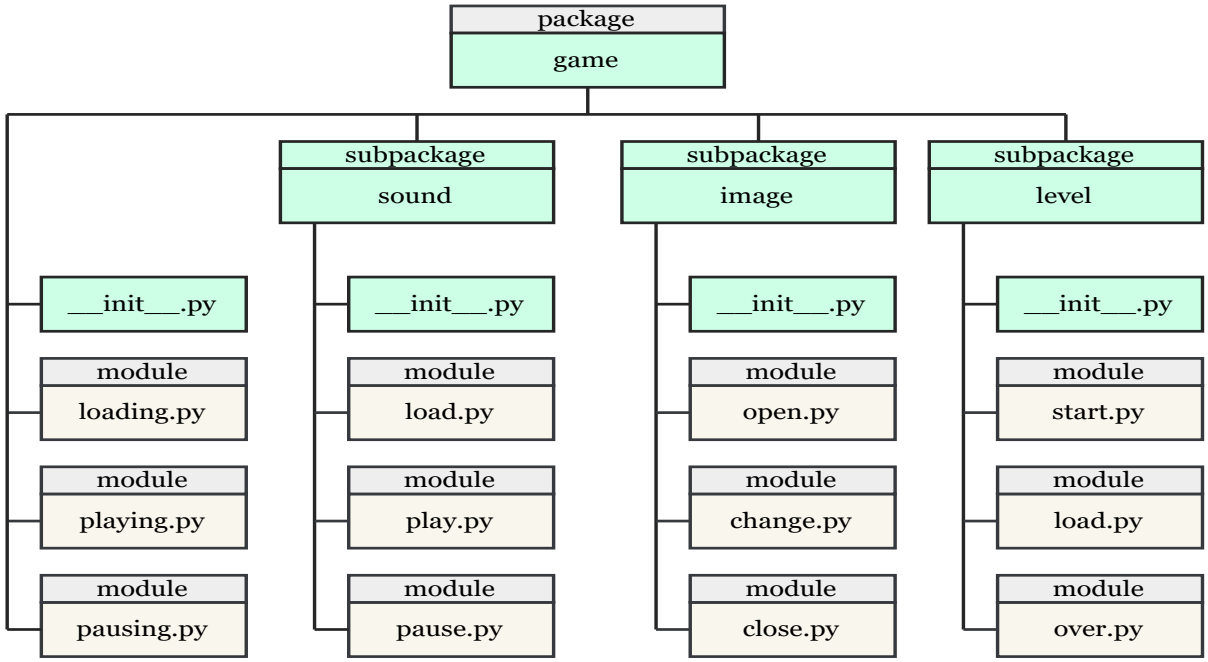
A. **Create a Python File:** Start by creating a new Python file with a `.py` extension. This file will contain the code for your module.

B. **Define Functionality:** Write the Python code that you want to include in your module. This can include function definitions, variable assignments, class definitions, and executable statements.

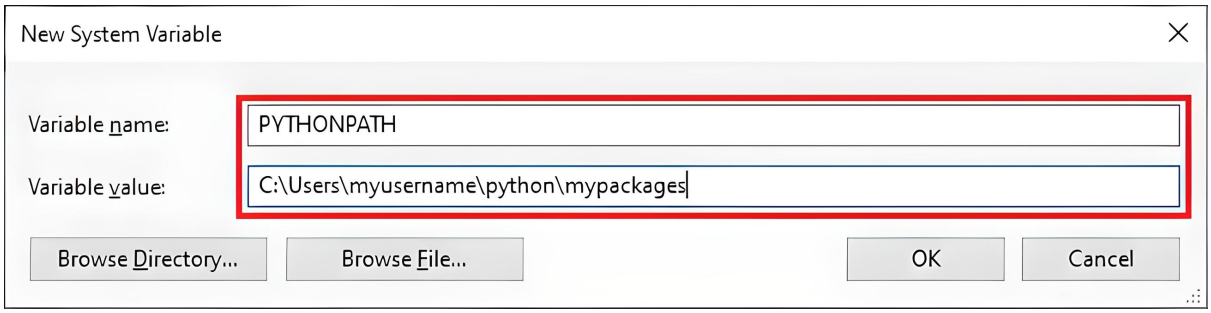
C. **Save the File:** Save the Python file with an appropriate name that reflects the functionality it provides. This name will be used as the name of the module when importing it into other code.

D. **Import and Use:** Once your module is created, you can import it into other Python programs or modules using the `import` statement. You can then use the functions, variables, and classes defined in the module as needed.

Python Packages: Python packages are basically a directory of a collection of modules. To be considered a package (or subpackage), a directory must contain a file named __init__.py. This file usually includes the initialization code for the corresponding package. Example of an own created python package - https://thingsdaq.org/2023/01/29/python-modules-and-packages/



Before we can start importing modules and their components, we have to make sure that the 'package' folder can be found by the Python interpreter. One good way to ensure that that is always the case is by adding it to the PYTHONPATH environment variable. Under the System variables, if there isn't already a PYTHONPATH variable, create a new one as shown next. Otherwise, you can add the package path to the existing one (separated by a semi-colon).



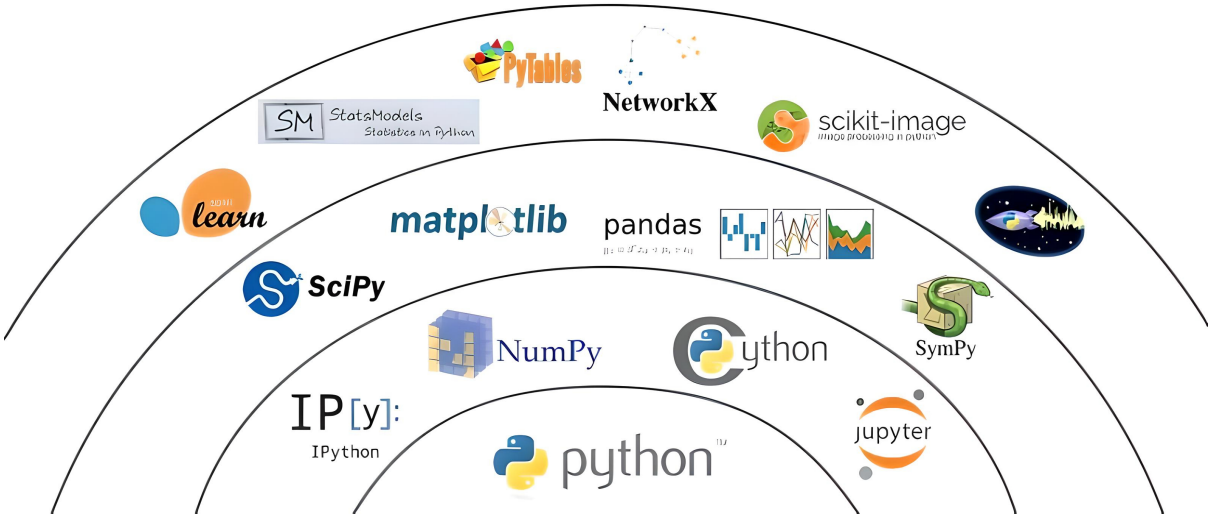
Calling module & package - optional 'as' keyword for alis them	Example Code

`import package`
`from package import module`
`from package.module import *`
`from package.module import myfun, myvar`
`from package.subpackage.module import myfun, myvar`
	`import game`
	`from game import loading`
	`from game.loading import *`
	`from game.playing import f1,v1`
	`from game.sound.play import f1,v1`

What about the packages that we can install using pip (preferred installer program)? They are also a collection of modules that can help us solve specific problems, neatly organized in self contained distributable packages. In my case, since I have administrator rights on a Windows platform running Python 3.9, pip installed packages end up in the folder C:Files39-packages. That location is automatically on the Python search path and doesn't need to be in the PYTHONPATH environment variable. Modules in those packages (such as numpy or scipy) are readily available for imports into your code. These more professional packages can be found in the PyPi (Python Package Index) public repository. In fact, anyone can create a package and upload it to the PyPi repository.

5.2.2 Group - The Libraries and Frameworks

Python Libraries: A library is an umbrella term referring to a reusable chunk of code. Usually, a Python library contains a collection of related modules and packages. Actually, this term is often used interchangeably with "Python package" because packages can also contain modules and other packages (subpackages). However, it is often assumed that while a package is a collection of modules, a library is a collection of packages. Example: Matplotlib , PyTorch , pygame , BeautifulSoup , Requests , missingno.



Python Frameworks: Similar to libraries, Python frameworks are a collection of modules and packages that help programmers to fast track the development process. However, frameworks are usually more complex than libraries. Also, while libraries contain packages that perform specific operations, frameworks contain the basic flow and architecture of the application. Example: Django, Flask, Bottle.

A. Make Classes for the Banking System

- **Account**: Attributes include account number, balance, account type. Methods include deposit(amount), withdraw(amount), and get_balance().
- **Customer**: Attributes include name, address, phone, accounts. Methods include open_account(), close_account(), and get_accounts().
- **Transaction**: Attributes include transaction type, amount, date, involved accounts. Methods include execute() and reverse().
- **Bank**: Attributes include name, address, customers, accounts. Methods include add_customer(), remove_customer(), and process_transaction().

B. Make Classes for the Library Management System

- **Book**: Attributes include title, author, ISBN, availability. Methods include check_out(), return_book(), and reserve().
- **Library**: Attributes include books, members, staff. Methods include add_book(), remove_book(), and find_book().
- **Member**: Attributes include name, membership ID, borrowed books. Methods include borrow_book(), return_book(), and pay_fines().

C. Make Classes for the Online Shopping System

- **Product**: Attributes include product ID, name, price, stock. Methods include add_to_cart(), remove_from_cart(), and update_stock().
- **ShoppingCart**: Attributes include items, quantities, customer. Methods include add_item(), remove_item(), and checkout().
- **Order**: Attributes include order ID, products, total amount, status. Methods include place_order(), cancel_order(), and track_order().
- **Payment**: Attributes include amount, method, status. Methods include process_payment(), refund(), and verify().

D. Make Classes for the Flight Reservation System

- **Flight**: Attributes include flight number, departure, arrival, seats. Methods include book_seat(), cancel_flight(), and update_schedule().
- **Passenger**: Attributes include name, passport number, bookings. Methods include book_flight(), cancel_booking(), and check_in().
- **Ticket**: Attributes include ticket ID, flight details, passenger info. Methods include issue_ticket(), change_flight(), and refund_ticket().
- **Airline**: Attributes include name, flights, staff. Methods include add_flight(), remove_flight(), and manage_bookings().

E. Make Classes for the School Management System

- **Student**: Attributes include student ID, name, enrolled courses, grades. Methods include enroll_course(), drop_course(), and receive_grade().
- **Teacher**: Attributes include teacher ID, name, assigned courses. Methods include assign_grade(), create_course(), and schedule_exam().
- **Course**: Attributes include course ID, title, students, syllabus. Methods include add_student(), remove_student(), and update_syllabus().
- **Grade**: Attributes include course, student, letter grade. Methods include set_grade(), get_grade(), and appeal_grade().

F. Make Classes for the Inventory Management System

- **Item**: Attributes include item ID, name, quantity, price. Methods include restock(), sell(), and check_inventory().
- **Inventory**: Attributes include items, quantities, location. Methods include add_item(), remove_item(), and audit().
- **Supplier**: Attributes include supplier ID, name, products. Methods include deliver(), receive_order(), and negotiate_contract().
- **Order**: Attributes include order ID, items, supplier, status. Methods include place_order(), receive_items(), and cancel_order().

G. Make Classes for the Hospital Management System

- **Patient**: Attributes include patient ID, name, medical history, appointments. Methods include schedule_appointment(), update_history(), and discharge().
- **Doctor**: Attributes include doctor ID, name, specialty, patients. Methods include diagnose(), prescribe_treatment(), and perform_surgery().
- **Appointment**: Attributes include appointment ID, patient, doctor, time. Methods include confirm(), reschedule(), and cancel().
- **MedicalRecord**: Attributes include record ID, patient details, treatments. Methods include add_entry(), update_entry(), and retrieve_record().

H. Make Classes for the Social Media Platform

- **User**: Attributes include user ID, name, profile, posts. Methods include post_content(), send_message(), and add_friend().
- **Post**: Attributes include post ID, content, author, comments. Methods include like(), comment(), and share().
- **Comment**: Attributes include comment ID, text, author, post. Methods include reply(), edit(), and delete().
- **Message**: Attributes include message ID, sender, receiver, content. Methods include send(), receive(), and delete().

I. Make Classes for the Real Estate Portal

- **Property**: Attributes include property ID, address, price, status. Methods include list_for_sale(), update_details(), and mark_sold().
- **Agent**: Attributes include agent ID, name, properties, clients. Methods include add_listing(), remove_listing(), and contact_client().
- **Listing**: Attributes include listing ID, property details, agent info. Methods include publish(), update_price(), and withdraw().
- **Inquiry**: Attributes include inquiry ID, customer, property, agent. Methods include submit(), follow_up(), and close().

J. Make Classes for the Game Development

- **Character**: Attributes include character ID, name, health, inventory. Methods include move(), attack(), and pick_up_item().
- **Level**: Attributes include level number, layout, enemies, items. Methods include load_level(), save_progress(), and spawn_enemy().
- **Enemy**: Attributes include enemy ID, type, health, behavior. Methods include patrol(), engage(), and drop_loot().
- **Item**: Attributes include item ID, name, effect, rarity. Methods include use(), equip(), and discard().

K. Make Classes for the Restaurant Ordering System

- **Menu**: Attributes include items, prices, specials. Methods include add_item(), remove_item(), and update_prices().
- **Order**: Attributes include order ID, menu items, table number. Methods include place_order(), modify_order(), and bill_customer().
- **Table**: Attributes include table number, capacity, status. Methods include reserve(), release(), and clean_up().
- **Waiter**: Attributes include waiter ID, name, assigned tables. Methods include take_order(), serve_food(), and process_payment().

L. Make Classes for the Car Rental Service

- **Vehicle**: Attributes include vehicle ID, make, model, availability. Methods include rent_out(), return_vehicle(), and service().
- **Rental**: Attributes include rental ID, customer, vehicle, duration. Methods include start_rental(), extend_rental(), and end_rental().
- **Customer**: Attributes include customer ID, name, rental history. Methods include rent_vehicle(), return_vehicle(), and pay_charges().
- **Agency**: Attributes include agency ID, name, fleet, locations. Methods include add_vehicle(), remove_vehicle(), and manage_rentals().

M. Make Classes for the Movie Ticket Booking System

- **Movie**: Attributes include movie ID, title, duration, ratings. Methods include schedule_showtime(), update_info(), and add_review().
- **Cinema**: Attributes include cinema ID, name, screens, showtimes. Methods include add_movie(), remove_movie(), and sell_tickets().
- **Showtime**: Attributes include showtime ID, movie, time, available seats. Methods include book_seat(), cancel_showtime(), and update_timing().
- **Ticket**: Attributes include ticket ID, showtime details, seat number. Methods include issue_ticket(), refund_ticket(), and change_seat().

N. Make Classes for the Fitness Club Management

- **Member**: Attributes include member ID, name, membership level, activities. Methods include sign_up(), book_class(), and cancel_membership().
- **Trainer**: Attributes include trainer ID, name, specialties, schedule. Methods include assign_workout(), track_progress(), and schedule_session().
- **Workout**: Attributes include workout ID, description, duration, intensity. Methods include start_workout(), modify_routine(), and log_results().
- **Equipment**: Attributes include equipment ID, type, status, maintenance records. Methods include check_out(), return_equipment(), and schedule_maintenance().

O. Make Classes for the Music Streaming Service

- **Song**: Attributes include song ID, title, artist, genre. Methods include play(), add_to_playlist(), and rate().
- **Playlist**: Attributes include playlist ID, name, songs, creator. Methods include add_song(), remove_song(), and share().
- **User**: Attributes include user ID, name, playlists, listening history. Methods include subscribe(), discover_music(), and follow_artist().
- **Subscription**: Attributes include subscription ID, user, plan, benefits. Methods include upgrade_plan(), renew(), and cancel_subscription().