Part I - Theoretical Python by Mostafizur Rahaman, Lecturer, ESE, KUET

Chapter 1 - The Computer: Hardware and Software

1.1 Hardware	Hardware = Input, Process (CPU), Output
1.2 Software	Software = System Software, Application Software

Chapter 2 - The Expression: Data and Operation

2.1 Data	var_name = data_value
2.2 Operation	var_name = data sign data

Chapter 3 - The Logic: Condition and Loop

3.1 Condition	Condition = if-elif-else, try-except-else-finally,with-as
3.2 Loop	Loop = for-else, while-else

Chapter 4 - The Process: Function and Concurrency

4.1 Function	Function = Builtin, Uniline User Defined, Multiline User Defined
4.2 Concurrency	Concurrency = Multithreading, MultiProcessing

Chapter 5 - The Box: OOP and Group

5.1 OOP	OOP = AM, EP, IA
5.2 Group	Group = Module_to_Package, Library_to_Framework

Part II - Applied Python [The Data Science] by Mostafizur Rahaman, Lecturer, ESE, KUET

Chapter 1 - Business Problem

1.1 Problem Identification	
1.2 Goal Definition	

Chapter 2 - Data Collection

2.1 Data Collection	
2.2 Data Preprocessing	

Chapter 3 - Feature Engineering

3.1 Nonsequential Data	
3.2 Sequential Data	

Chapter 4 - Model Training

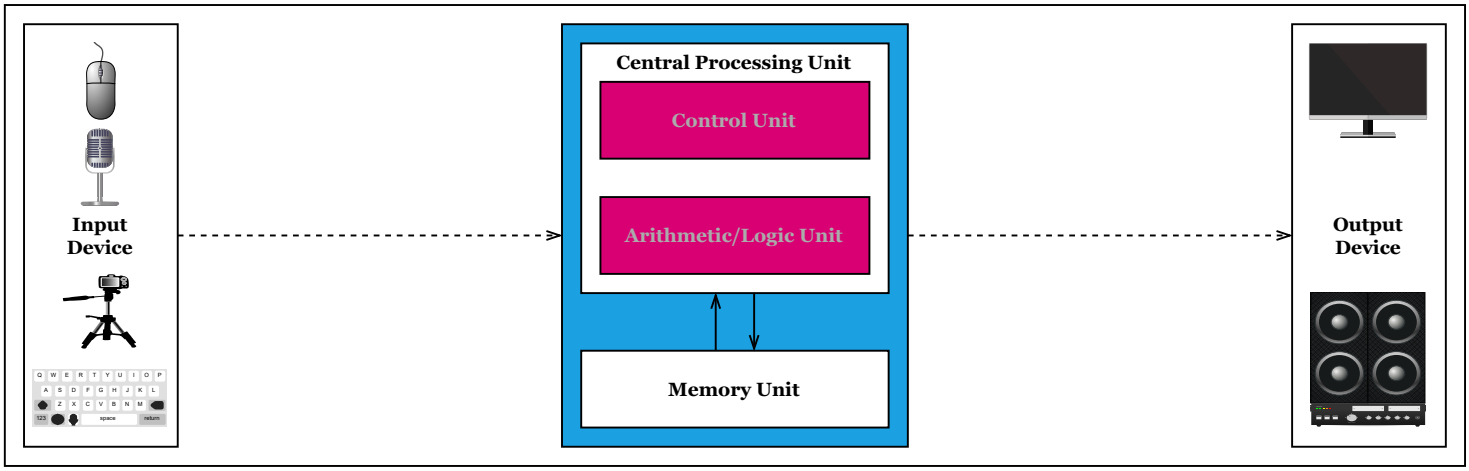
4.1 Machine Learning (ML)	
4.2 Deep Learning (DL)	

Chapter 5 - Model Evaluation

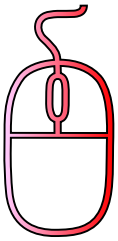
5.1 Regression Metrics	
5.2 Classification Metrics	

Chapter 1 - The Computer: Hardware and Software

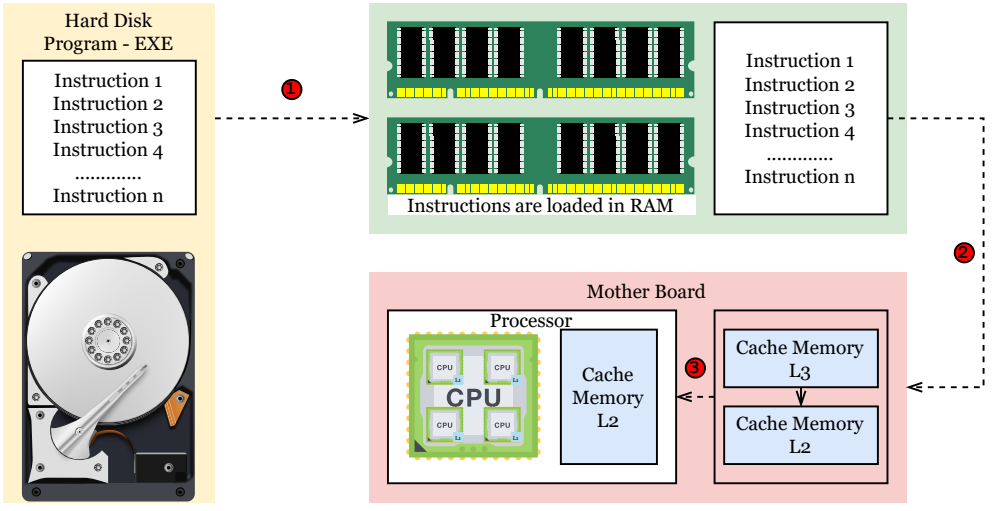
1.1 Hardware



1.1.1 Input Devices

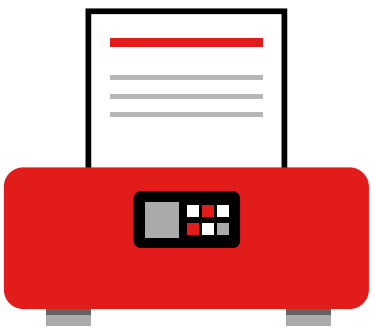


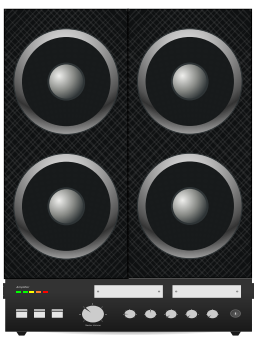
1.1.2 Central Processing Unit



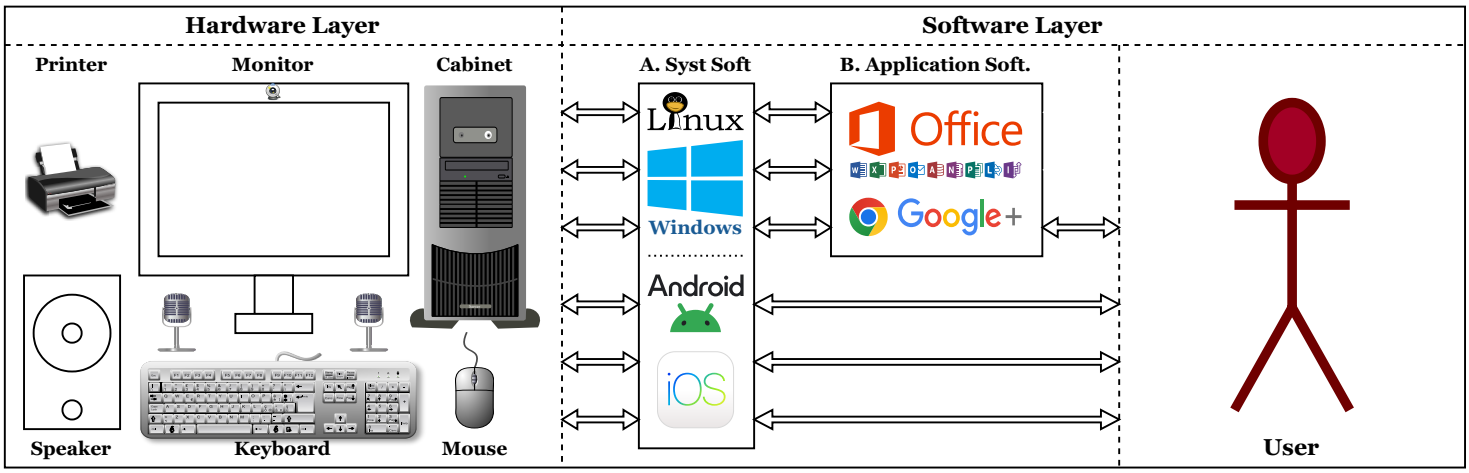
RAM	
Address	Value
0000	0101 0101
0001	0001 0110
0010	0011 0111
0011	1001 0010
0100	0000 0000
0101	0011 0000
0110	0000 0001
0111	0000 0000
1000	0000 0000
1001	0000 0000
1010	0000 0000
1011	0000 0000
1100	0000 0000
1101	0000 0000
1110	0000 0000
1111	0000 0000

1.1.3 Output Devices





1.2 Software



1.2.1 System Software

▶ Operating System → Command Prompt → Power Shell

1.2.1 Application Software

▶ Python.exe → Anaconda.exe/VSCode.exe (Jupyter Notebook) → Learn Language

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
```

OOP Functions:

Object lifecycle

```
__new__()
__init__()
__del__()
__init_subclass__()
__set_name__()
__prepare__()
```

String representation

```
__repr__()
__str__()
__bytes__()
__format__()
```

Type conversions

```
__bool__() __complex__()
__int__() __float__()
__index__()
```

Comparisons

```
__lt__()
__le__()
__eq__()
__ne__()
__gt__()
__ge__()
```

Type check

```
__instancecheck__()
__subclasscheck__()
```

Context manager

```
__enter__()
__exit__()
```

Attribute access

```
__getattr__()
__getattribute__()
__setattr__()
__delattr__()
__dir__()
__get__()
__set__()
__delete__()
__slots__()
```

Operations on sequences

```
__len__()
__length_hint__()
__getitem__()
__setitem__()
__delitem__()
__missing__()
__iter__()
__next__()
__reversed__()
__contains__()
```

Math operations

```
__add__() __sub__() __mul__() __matmul__()
__truediv__() __floordiv__() __mod__() __divmod__()
__pow__() __lshift__() __rshift__() __and__()
__xor__() __or__()

__radd__() __rsub__() __rmul__() __rmatmul__()
__rtruediv__() __rfloordiv__() __rmod__()
__rdivmod__() __rpow__() __rlshift__() __rrshift__()
__rand__() __rxor__() __ror__()

__iadd__() __isub__() __imul__() __imatmul__()
__itruediv__() __ifloordiv__() __imod__() __ipow__()
__ilshift__() __irshift__() __iand__() __ixor__()
__ior__()

__neg__() __pos__() __abs__() __invert__()
__round__() __trunc__() __floor__() __ceil__()
__hash__()
```

Asynchronous

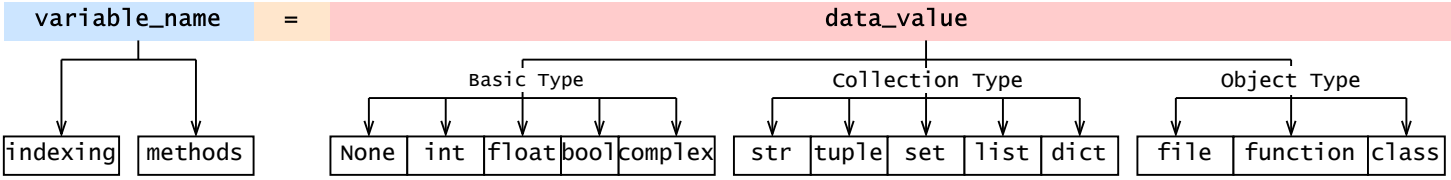
```
__await__()
__aiter__()
__anext__()
__aenter__()
__aexit__()
```

Miscellaneous

```
__call__()
__class_getitem__()
__mro_entries__()
```

Chapter 2 - The Expression: Data and Operation

2.1 Data



2.1.1 data_value

Data Types		Definition and Example	Properties			
			Indexed	Ordered	Changeable	Duplicate
Basic Types	NoneType int float complex bool	a = None b1 = 34, b2=-55, b2=0b11, b3=0o65,b4=0x9A z1 = 3.1415, z2 = -6.25 d = -3 + 4j e1 = True, e2 = False	-	-	-	-
Collection Types	str bytes bytearray memoryview	m1 = 'Hello World', m2 = "World's Mine"	☑	☑	☒	☑
	tuple range	n1 = ("apple", "banana", "cherry") n2 = tuple(("apple", "banana", "cherry"))	☑	☑	☒	☑
	set fresenset	o1 = {"apple", "banana", "cherry"} o2 = set(("apple", "banana", "cherry"))	☒	☒	☒	☒
	list	p1 = ["apple", "banana", "cherry"] p2 = list(("apple", "banana","cherry"))	☑	☑	☑	☑
	dict	q1 = {"name": "John", "age": 36} q2 = dict(name="John", age=36)	☒	☑	☑	☒
User Defined Types	File Function Class	x = open('filename.mp3', 'rb') y = lambda a,b,c: a*b**c class myclass: z = 5	-	-	-	-

2.1.2 var_name[start:end:step]

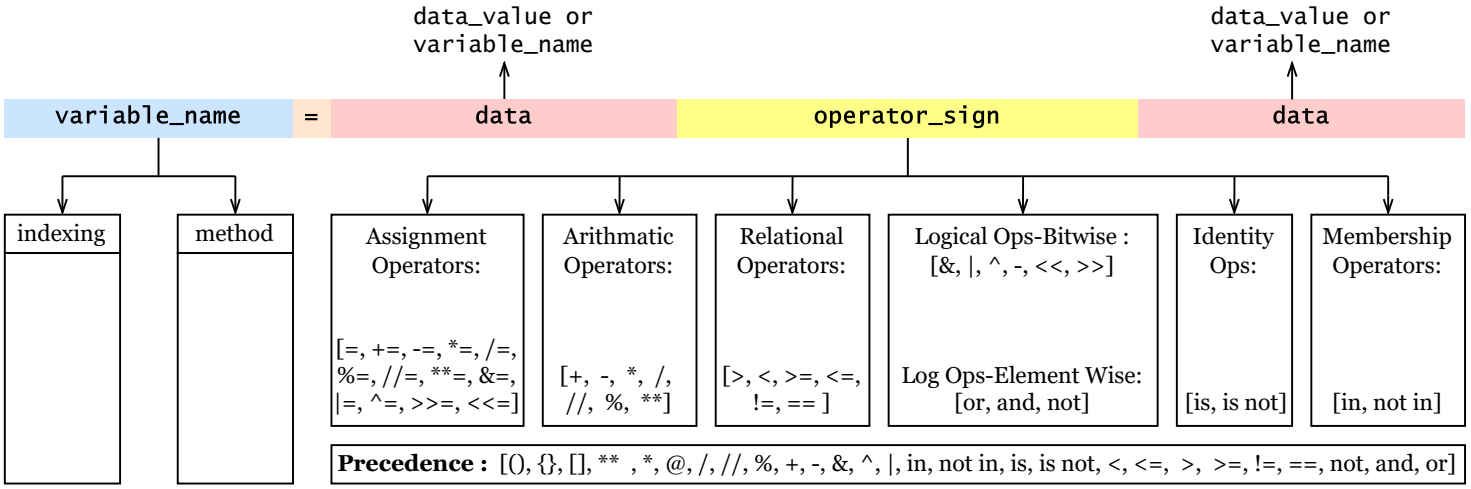
```
word = "amazing"

print(word[0:4])      # amaz
print(word[:5])       # amazi
print(word[1:5:2])    # mz
print(word[-7:-3])    # amaz
print(word[-5:])      # azing
print(word[-5:-1:2]) # ai
print(word[::-2])     # gzm
```

```
my_list = [10, 20, 30, 40, 50, 60, 70]

print(my_list[0:4])      # [10, 20, 30, 40]
print(my_list[:5])       # [10, 20, 30, 40, 50]
print(my_list[1:5:2])    # [20, 40]
print(my_list[-7:-3])    # [10, 20, 30, 40]
print(my_list[-5:])      # [30, 40, 50, 60, 70]
print(my_list[-5:-1:2]) # [30, 50]
print(my_list[::-2])     # [70, 50, 30, 10]
```

2.1.3 var_name.method_name(input_01, . .,input_on)

☒ NoneType Methods List: -
☒ int Methods: as_integer_ratio, bit_count, bit_length,conjugate, denominator, from_bytes, imag, numerator, real, to_bytes
☒ float Methods: as_integer_ratio, conjugate, fromhex, hex, imag, is_integer, real
☒ complex Methods: conjugate, imag, real
☒ bool Methods: as_integer_ratio, bit_count, bit_length, conjugate, denominator, from_bytes, imag, numerator, real, to_bytes
☒ str Methods: 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.
☒ tuple Methods: count, index
☒ set Methods: add, clear, copy, difference, difference_update, discard, intersection, intersection_update, isdisjoint, issubset, issuperset, pop, remove, symmetric_difference, symmetric_difference_update, union, update
☒ list Methods: append, clear, copy, count, extend, index, insert, pop, remove, reverse, sort
☒ dict Methods: clear, copy, fromkeys, get, items, keys, pop, popitem, setdefault, update, values

2.2 Operations



2.2.1 Simple Operation

```
x, y = 5, 3          # x = 5; y = 3
num_res_01 = x + y   # Addition
num_res_02 = x - y   # Subtraction
num_res_03 = x * y   # Multiplication
num_res_04 = x / y   # Division
num_res_05 = x // y  # Floor Division
num_res_06 = x % y   # remainder
num_res_07 = x ** y  # exponentiation
```

```
x, y = 5, 3          # x = 5; y = 3

com_res_01 = x < y   # Less than
com_res_02 = x == y  # Equal to
com_res_03 = x > y   # Greater than
com_res_04 = x != y  # Not equal to
com_res_05 = x >= y  # Greater than or equal
com_res_06 = x <= y  # Less than or equal
```

```
x, y = True, False



log_res_01 =x and y
log_res_02 =x or  y
log_res_03 =  not x
```

2.2.2 Complex Operation

Mathematical Calculation

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: $6 \times (2 - 1) = 6$

Python Code

```
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
```

Chapter 3 - The Logic: Condition and Loop

3.1 The Conditional Control Statement

3.1.1 if-elif-else Cond Cont Stat 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")
```

Application

3.1.2 try-except-else-finally Cond Cont Stat Synt

```
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")
```

Application

3.1.3 with-as Context Manager Stat 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)
```

Application

3.2 The Loop Control Statement

3.2.1 for-else Loop Control Statement Syntax

```
for target_var in data: # 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)
```

Application

Advanced for loop 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)
```

Output

```
1 11 21
2 12 22

1 apple
2 banana
3 orange
```

Application

Advanced for loop example code

```
# 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}
```

Application

3.2.2 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
```

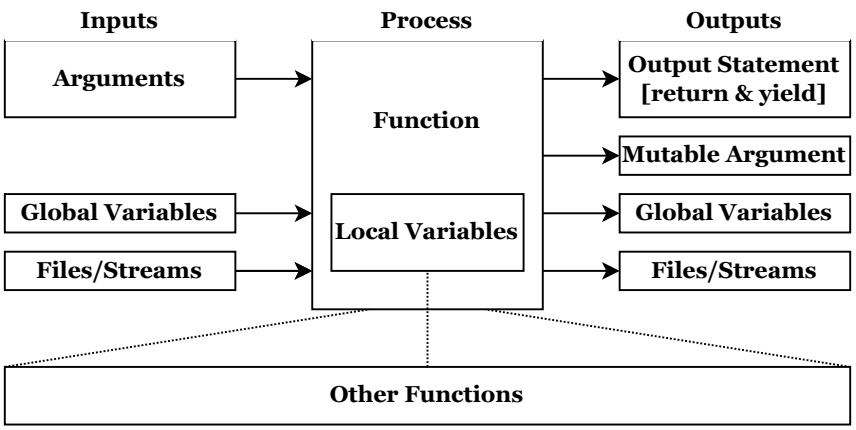
Example Code

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

Application

Chapter 4 - The Process: Function and Concurrency

4.1 Function



Built-in (Python)
Names preassigned in the built-in names module: open, range, SyntaxError ... .

Global (module)
Names assigned at the top-level of a module file, or declared global in a def within the file.

Enclosing function locals
Names in the local scope of any and all enclosing functions (def or lambda), from inner to outer.

Local (function)
Names assigned in any way within a function (def or lambda), and not declared global in that function.

4.1.1 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

```
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))
```

4.1.2 Unline 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)
```

4.1.3 Multiline 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)
```

Advanced argument matching python syntax

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

Example Code

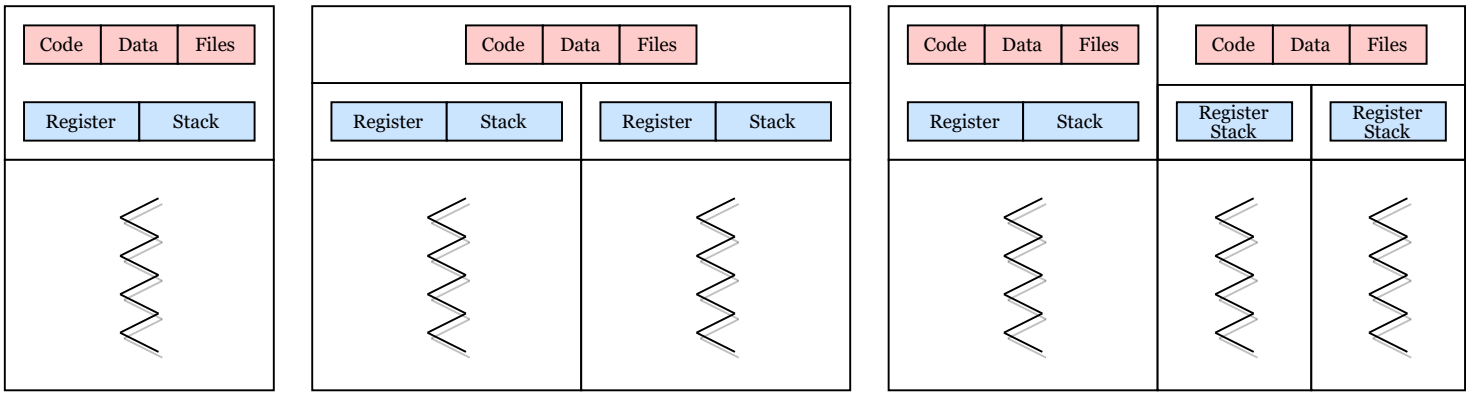
```
def fun10(a, b, c=0, *u, **v):
    print("a (normal  arg):",a)
    print("b (normal  arg):",b)
    print("c (default arg):",c)
    print("v (dict    arg):",v)
fun10(1, 2, 3, 4, 5, x=6, y=7)
```

Scope of variable

```
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)
```

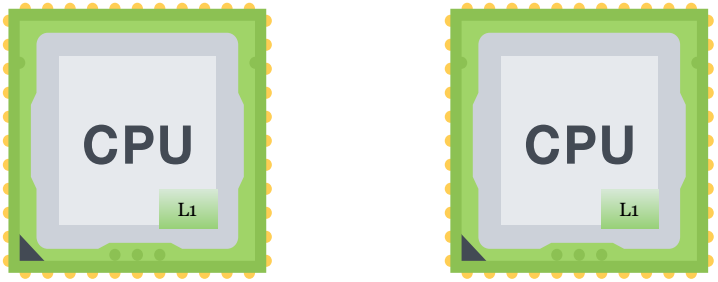
Application

4.2 Concurrency



A. Multithreading: Concurrency as Not Parallel - a. Single Threading b. Multithreading

B. Multiprocessing: Concurrency as Parallel



4.2.1 Multithreading

Multithreading Syntax

```
from threading import Thread

def task1_part1(arg1, arg2):
    print(f"Task 1 Part 1: {arg1}, {arg2}")

def task1_part2(arg):
    print(f"Task 1 Part 2: {arg}")

def task2(arg1, arg2, arg3):
    print(f"Task 2: {arg1}, {arg2}, {arg3}")

t1 = Thread(target=task1_part1, args=("Hi", "By"))
t2 = Thread(target=task1_part2, args=("Python",))
t3 = Thread(target=task2, args=(1, 2, 3))

t1.start()
t2.start()
t3.start()

t1.join()
t2.join()
t3.join()
```

Multithreading Example

```
import threading
import time


def download_file(file_name, duration):
    print(f"Starting download of {file_name}...")
    time.sleep(duration)
    print(f"Finished download of {file_name}!")

thread1 = Thread(target=download_file, args=("file1.txt", 3))
thread2 = Thread(target=download_file, args=("file2.txt", 5))
thread3 = Thread(target=download_file, args=("file3.txt", 2))


thread1.start() # Function that simulates downloading a file
thread2.start() # ----------------------------------------
thread3.start() # ----------------------------------------


thread1.join()  # Waiting for all threads to finish
thread2.join()  # --------------------------------
thread3.join()  # --------------------------------

print("All downloads completed!")
```

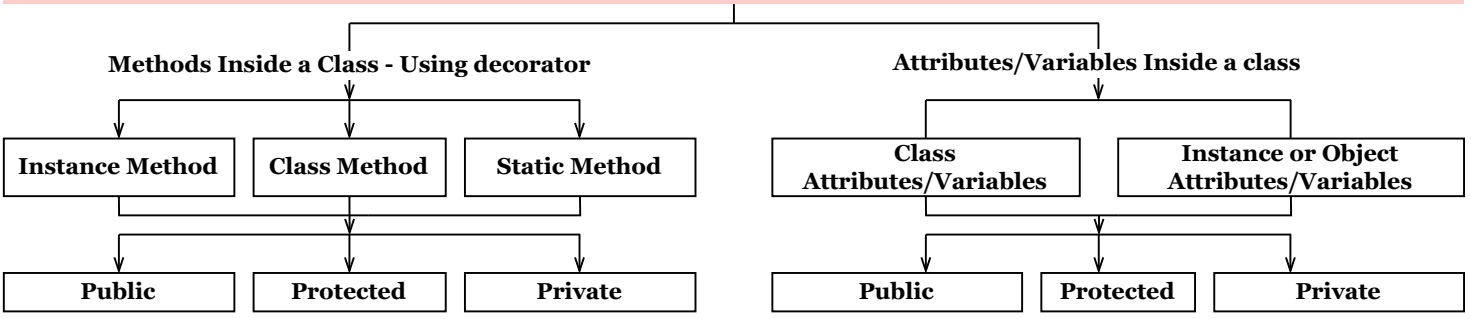
4.2.2 Multiprocessing

Chapter 5 - The Box: OOPs and Groups

class_identifier class_name:	class bank:	class student:	class book:	class circle:
Data Members (Static Attributes) Data Members (Static Attributes) --- ---	name location branches ---	name grade --- ---	title author isbn_no ---	radius colors --- ---
Function Members (Dynamic Operations) Function Members (Dynamic Operations) --- ---	get_branch() get_location() --- ---	getname() printgrade() --- ---	get_title() give_author() --- ---	get_radius() give__area() --- ---

5.1 OOP

5.1.1 Attributes and Methods ≡ @property, @staticmethod, @classmethod



5.1.2 Encapsulation and Polymorphism

Encapsulation ≡ Public, Protected, Private

Polymorphism

Object lifecycle

```
__new__()
__init__()
__del__()
__init_subclass__()
__set_name__()
__prepare__()
```

String representation

```
__repr__()
__str__()
__bytes__()
__format__()
```

Type conversions

```
__bool__() __complex__()
__int__() __float__()
__index__()
```

Comparisons

```
__lt__()
__le__()
__eq__()
__ne__()
__gt__()
__ge__()
```

Type check

```
__instancecheck__()
__subclasscheck__()
```

Context manager

```
__enter__()
__exit__()
```

Attribute access

```
__getattr__()
__getattribute__()
__setattr__()
__delattr__()
__dir__()
__get__()
__set__()
__delete__()
__slots__()
```

Operations on sequences

```
__len__()
__length_hint__()
__getitem__()
__setitem__()
__delitem__()
__missing__()
__iter__()
__next__()
__reversed__()
__contains__()
```

Math operations

```
__add__() __sub__() __mul__() __matmul__()
__truediv__() __floordiv__() __mod__() __divmod__()
__pow__() __lshift__() __rshift__() __and__()
__xor__() __or__()

__radd__() __rsub__() __rmul__() __rmatmul__()
__rtruediv__() __rfloordiv__() __rmod__()
__rdivmod__() __rpow__() __rlshift__() __rrshift__()
__rand__() __rxor__() __ror__()

__iadd__() __isub__() __imul__() __imatmul__()
__itruediv__() __ifloordiv__() __imod__() __ipow__()
__ilshift__() __irshift__() __iand__() __ixor__()
__ior__()

__neg__() __pos__() __abs__() __invert__()
__round__() __trunc__() __floor__() __ceil__()
__hash__()
```

Asynchronous

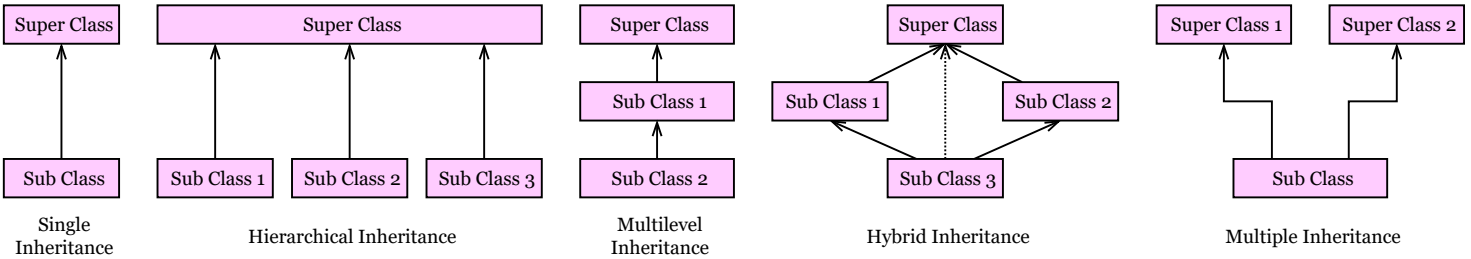
```
__await__()
__aiter__()
__anext__()
__aenter__()
__aexit__()
```

Miscellaneous

```
__call__()
__class_getitem__()
__mro_entries__()
```

5.1.3 Inheritance and Abstraction

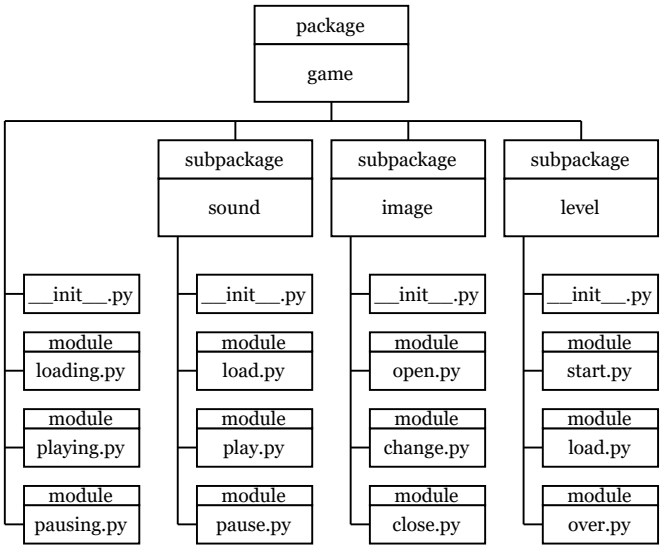
Inheritance



Abstraction ≡ @abstractmethod

5.2 Group

5.2.1 Module_to_Package



mymath1.py

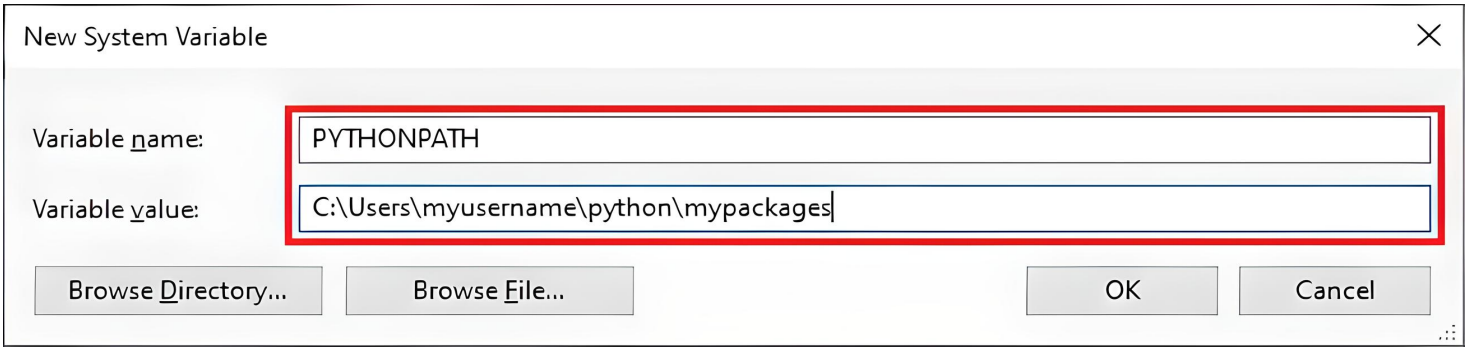
```
def addition(x, y):
    """sum"""
    return x + y

def subtraction(x, y):
    """difference"""
    return x - y

def multiplication(x, y):
    """product"""
    return x * y

def division(x, y):
    """division"""
    return x / y
```

mymath2.py

```
class Calculator:

    def addi(self, x, y):
        return x + y

    def subt(self, x, y):
        return x - y

    def mult(self, x, y):
        return x * y

    def divi(self, x, y):
        if y == 0:
            raise ValueError("Not")
        return x / y
```

```
import mymath1


x = mymath1.addition(5, 3)
```

```
from mymath2 import Calculator

calc   = Calculator()
result = calc.addi(5, 3)
```



5.2.2 Library_to_Framework