Introduction to Python

- high level, interpreted programming language known for its simplicity and readability. It supports multiple programming paradigms, such as object-oriented, procedural and functional programming

Programme Development Cycle

→ Analysing the problem → designing the problem → coding → testing and debug. → deployment → maintainance

Identifiers: names given to variables, functions, classes etc.
- → must start with letter or underscore
- → can contain number, cannot contain keyword

Keywords: Reserved words in Python that have special meaning
- → if, else, for, while, input etc.

Statements: An instruction that the Python interpreter can execute,
- → print or if

Expressions: combination of variables, values and operators that Python interprets and computes to produce another value.
- → logical (and, or, not)
- → comparisn (==, <=, >=)

Variables: It is a named location in memory that stores data that can be used and manipulated in the program
- → need not declare variable type, it is dynamically assigned. (integer, float, string, boolean)
- → case sensitive

Operators: operators perform operations on variable & values.

1. ARITHMETIC 2. LOGICAL 3. RELATIONAL

1/0 ⇒ error
True or 1/0 ⇒ True
1/0 or True ⇒ error
} SHORT CIRCUIT EVALUATN

Page No.
Date

Precedence and Associativity

→ Operator precedence determine the order of operations in expression.

→ Associativity determines the direction (R→L or L→R) when 2 operators of the same precedence are evaluated.

(**) ⟶ (+x, -x) ⟶ (*, /, //, %) ⟶ (+ -)

→ Python evaluates expressions from left to right

→ 2 exceptions : ** and = : evaluted from right to left.

not ⟶ and ⟶ or : evaluated left to right.

Data Types.

int: integer values, float: floating-point values,
str: string values, bool: boolean values

Decimal form: base 10
→ default no. system - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Binary form: base -2
→ permitted values 0 or 1, literal values prefixed with 0b or 0B.

Octal system: base -8
→ permitted values - 0, 1, 2, 3, 4, 5, 6, 7, literal values are prefixed with 0o or 0O

Hexadecimal form: base -16
→ permitted values - 0, 1, 2, 3, 4, 5, 6, 7, 8, a-f; literal values are prefixed with 0x or 0X

string slicing: St [: :]
start, end, step

reverse string: st [: : -1]

Indentation

(loops or condn)

Python uses indentation to define code blocks,

Comments

Comments are used to explain codes. Single-line comments starts with #, multiline comments use triple quotes """ or ''' '''

Input: input ("Enter:")

Output: print ("word")

Type conversion

→ int(), float(), str() to convert data types.

Decision Structures and Boolean Logic

- if: test condition & execute code only if condn is TRUE.
- if-else: executes one block of code if condn is True, executes other block if condn is False.
- if-elif-else: allow checking of multiple conditions
- Nested decision structure: if statements inside if

comparing strings

relational operators: ==, !=, <, >, >=, <=

Logical operators

and: True if both condn are True
or: True if any one condn is True
not: reverses the Truth value.

Boolean variables

variables that store True & False.

Repitition structures

while loop: continues as long as condition remains True.
for loop: used to iterate over a sequence (list, tuple, str, range)
nested loop: loop inside a loop.

break: exits the loop entirely
continue: skips the current iteratn & moves to next one.

Strings

- accessing characters and substrings, string slicing. (Indexing)
  str[5] → specific index
  str[i : j : k] → start, end, step
- string and number system
  → int() and float() : convert no. str to numeric type
- string methods
  → st.upper() : converts all to upper case
  → st.lower() : converts all to lower case
  → st.strip() : removes both leading & trailing whitespaces
  → st.rstrip() : removes trailing white spaces
  → .split() : splits a string into list of substrings
  → .title() : all first characters are capitalized.
  → .capitalize : capitalizes the first letter of string.
  → st.index('str') : finds the index of substring.
  → .isalnum() : True if numeric & alphabet character
  → .is_alpha() : checks if all char are alphabet
  → .isdigit() : checks if all char are numeric
  → .isspace() : check if only whitespaces in string.
  → .istitle() : checks if string was title case.
  → .join() : joins string of char with argument.
  → .count() : returns no. of occurance of substring
- Basic String manipulation operations
  concatenation : + (add)
  repitition : * (multiply)
- Searching and manipulating string
  → .find() : finds lowest index of substring present.
  → .replace(old, new) :
  → .partition() : returns tuple of before partith, seperator, after partith.

Lists

- ordered, mutable collection of elements. []
- can store multiple data types.

- List slicing [start : end : step]
- Basic list operations
  - → .append('element') : appends item to end of list
  - → .insert('index', 'item') : inserts at a given inde
  - → remove(), del() : delete list / list item
- Functions for list
  - → len() : length of a list
  - → min(), max() : min, max value in list
  - → sum() : sum of all elements
  - → .extend('list') : add elements of given lst to Og.
  - → .pop() : remove last item & returns it
  - → .clear() : empties the list
  - → .reverse() : reverses the items of the list
  - → .sort() : sorts in ascending order.
  - → .sorted() : returns A NEW sorted list.
- Copying
  - .copy() : to avoid modifying orgn list

2D list

matrix = [[1,2,3], [4,5,6], [7,8,9])

Tuples

- immutable sequence in Python ()

- Tuple slicing [start : end : step]
- Basic tuple operations
  - concatenation : + (add) t1+t2
  - repetition : * (multiply) t1* (no)
- Built in function
  - → len()
  - → min(), max()
  - → sum()
  - → t[::-1] : reversing
  - → del(t) : deletes the whole tuple
  - → .count('value') : no. of times the value occured
  - → .index (x) : specific index of the value.

Sets

- unordered collection of unique elements {}

- Adding or removing elements
  - → .add(<value>) : adds new element.
  - → .remove(<val>) : removes element. ERROR if not found
  - → .discard(<val>) : removes element. NO ERRORS raised.
  - → .pop() : returns the last element.
- set operation
  - → S1.union(S2) = S1 ∪ S2
  - → S1.intersection(S2) = S1 ∩ S2
  - → S1.~~outer~~ difference(S2) = S1 - S2
  - → S1.symmetric_difference(S2) = (S1 ∪ S2) - (S1 ∩ S2)
- Frozenset

  sf = frozenset({"a", "b", "c"})
- Transversing → use for loop

Dictionary

— collection of key-value pair, ~~defined~~

- Accessing key-value pair
  dictionary [<key>]
- Modifying key-value pair
  dictionary [<key>] = <value>
- Built in functions
  - → len():
  - → ~~key~~ min(), max():
  - → sum(): sum(d1.values)
  - → .keys(): returns a list of keys
  - → .values(): returns a list of values.
  - → .items(): returns a list of key-value tuple.
  - → .fromkeys(keys, <var>): assigns same value to all keys in seq^
  - → .setdefault(key:value): inserts a new key value pair
- Dictionary methods.
  - → .get(<key>): returns the value of key
  - → .pop(): returns & remove the last ~~element~~ element.
  - → .popitem(): removes & returns the last key value.
  - → .copy(): creates copy of dictionary
  - → .sorted(): returns a sorted list of keys.
  - → .update(d2): inserts (key value) pair to dicts.
  - → .clear(): empties the dictionary

FACTORIAL

```
n = int(input("Input number:"))
f = 1
if n<0:
    print("enter positive number")
elif n==0:
    print("factorial of zero is 1")
else:
    for i in range(1,n+1):
        f = f*1
    print("Factorial of ",n,"is:",c)
```

PRIME NUMBER

```
num = int(input("Enter a number:"))
for i in range(2,n//2):
    if num % i==0:
        print("number is not prime")
        break
else:
    print("no. is prime")
```

ARMSTRONG NUMBER

```
num = int(input("Enter a number:"))

sum = 0
for i in num:
    sum = sum + int(i) ** len(num)
if int(num) == sum:
    print("ARMSTRONG NO.")
else:
    print("Not Armstrong no")
```

Page No.
Date

PALINDROME

```
st = input ("Enter string :"))
if st == st [: :-1]
    print ("PALINDROME")
else :
    print ("NOT A PALINDROME")
```

```
num = int (input ("Enter a num:"))
k = num
rev = 0
while (n > 0) :
    rem = num % 10
    num = num // 10
    rev = (rev * 10) + r
if k == rev :
    print ("PALINDROME")
else :
    print ("NOT A PALINDROME")
```

SUM OF DIGITS

```
num = int (input ("Enter a number :"))
sum = 0
while n != 0 :
    dig = n % 10
    sum = sum + dig
    n = n // 10
print ("Sum of digits is,", sum)
```

SORT WITHOUT SORTING:

```
st = input("Enter string:")
st = list(st)
for i in range(len(st)):
    for j in range(len(st)-i-1):
        if st[j] > st[j+1]:
            st[j], st[j+1] = st[j+1], st[j]
st_sorted = ""
for i in st:
    st_sorted += i
print(st_sorted)
```

(PYQ questn)

```
l = input("letter")
start = ord('A')
end = ord(l.upper())
for i in range(end-start+1):
    print(' '*i, end=' ')
    for j in range(start, end-i+1):
        print(chr(j), end='')
    for j in range(end-i-1, start-1, -1):
        print(chr(j), end=' ')
    print()
```

Output →

```
ABCDEFEDCBA
 ABCDEDCBA
  ABCDCBA
   ABCBA
    ABA
     A
```

PRIME FACTORS

```
def prime_factors(n):
    factor = 2
    factors = set()
    while n % factor == 0:
        factors.add(factor)
        n = n // factor
    factor = 3
    while factor * factor <= n:
        while n % factor == 0:
            factors.add(factor)
            n = n // factor
        factor = factor + 2

    if n > 2:
        factors.add(factor)

    return sorted(factors)

n = int(input("enter no."))

if n <= 1:
    print("enter valid no. greater than 1")
else:
    print("Prime factors of", n, "are:", prime_factors(n))
```

```
nterms = int(input("How many terms? "))
n1 = 0          # first 2 terms
n2 = 1
count = 0
if nterms < 0 :
    print("Enter positive integer")
elif nterms = 1 :
    print("Fibonacci sequence:")
    print(n1)
else :
    print("Fibonacci sequence:")
    while count < nterms :
        print(n1)
        nth = n1 + n2
        n1 = n2          } # update values
        n2 = nth
        count = +1
```

#

```
n = int(input("input number :"))
for i in range(1, n+1):
    for j in range(1, i+1):
        if j != i:
            print(j, end=',')
        else:
            print(j, end="")
    print()

for i in range(n-1, 0, -1):
    for j in range(1, i+1):
        if j != i:
            print(j, end=',')
        else:
            print(j, end="")
    print()
```

Output →

1
1,2
1,2,3
1,2,3,4
1,2,3,4,5
1,2,3,4
1,2,3
1,2.
1

decimal to 1. binary 2. octal 3. hexadecimal

```
decimal = int(input("Input value:"))
Binary = bin(decimal)
Octal = oct(decimal)
Hexadecimal = hex(decimal)
```

Functions

- blocks of reusable code that perform specific tasks. They help make code more organised, modular, and reusable.

Built-in functions: functions provided by Python
user defined funct^n: functions created by the programmer

```
def function_name (parameters):
    <function body>
    return value
```

- Function Parameters

* default parameters: allow a parameter to have a default value if no argument is provided. def funct(a=2, b=3)
* keyword arguments: allow specifying arguments by name, which helps readability.
  def funct (a, b)
  → funct (b=3, a=1)
* positional argument: arguments that need to be included in the proper order.
  funct (2, 3)
  assigns a=2 and b=3
* Variable-length arguments: used when number of positional arguments is unknown.
  *args: accepts any number of positional argument
  **kwargs: accepts any number of keyword arguments

Return : doesn't return any val rather it returns the control to caller along with empty value None.

return val : return val is returning the control to caller along with the value contained in variable.

VOID FUNCTN doesn't return val.

- Scope and lifetime of variables

Local variable
- variables declared within a function block
- cannot be accessed outside the function but only within a function / block of a program.

Global variable
- variable declared outside all the functions or in a global space
- accessible throughout the program in which it is declared.

- Commonly used modules

math module : provides mathematical functn and constants

random module : generates random module

OS and os.path module : os module provides functions for interacting with operating system, while os.path works with file paths.

date and time module : used for date and time manipulation.

scipy module : used for scientific computations. Requires installation via pip install scipy.

- Command line Arguments
  - inputs provided to a program when it is executed from command line.
- They allow users to pass values to a script without hardcoding them.
- sys module provides access to these arguments, which are stored in a list called sys.argv.

sys.argv is a list where
- sys.argv[0] is the script name
- sys.argv[1] and onwards are the additional arguments provided by the user.

example:

```
import sys                                  ← create a script greet.py
if len(sys.argv) > 1:
    name = sys.argv[1]
    print(f"Hello, {name}!")
else:
    print("Hello, World!")
```

python greet.py Alice → Output → Hello, Alice!

python greet.py → Output → Hello, World!

```
import sys
num1 = int(sys.argv[1])
num2 = int(sys.argv[2])
result = num1 + num2
print(f"Sum is: {result}")
```

python add.py 5 10 → Output → Sum is: 15

Recursion

- it is a programming technique where a function calls itself to solve a problem. Each recursive call solves a smaller part of the problem until it reaches a base case that stops the recursion.

1. Recursive case: the part of the function where it calls itself.
2. Base case:

- why use Recursion
  - → divide and conquer problems
  - → Simplified code

example: # Factorial Calculation

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

- Problem solving with Recursion
  1. identify the Base Case: determine the simplest case that stops the recursion.
  2. Define the Recursive Case: Formulate how the function calls itself with a smaller subproblem.
  3. Combine Results.

FIBONACCI SEQUENCE

$$F(n) = F(n-1) + F(n-2)$$

$$\begin{cases} F(0) = 0 \\ F(1) = 1 \end{cases}$$

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

SUM OF DIGITS

```
def sum_of_digits (n)
    if n < 10:
        return n
    else:
        return n % 10 + sum_of_digits (n // 10)
```

Files

Types

1. TEXT FILES : Stores data in a human-readable format with characters. (.txt files) They use encoding (eg UTF-8) to represent data.

2. BINARY FILES : stores data in binary format (like .jpg, .bin). Used for images, videos, & executable files.

TEXT FILES

CREating and Reading Text File

```
with open ("abc.txt", "w") as f :
        ("HELLO")

with open ("abc.txt", "r") as f :
data =
print (data)
```

File Methods to Read and write

⇒ WRITING

- write (<string>) : writes string to the file
- writelines (<data str>) : writes a list, tuple or dictionary to the file without adding newline character.

⇒ READING

- read (<size>) : reads a specified no. of characters
- readline ( ) : reads a single line
- readlines ( ) : Reads all lines and returns them in a LIST.

BINARY FILES

Reading and Writing Binary Files

```
with open ("svt.bin", "wb") as f:
f.write ("This is binary data")

with open ("svt.bin", "rb") as f:
data = f.read()
print (data)
```

pickle module: allows you to serialize (convert to byte stream) and deserialize (convert back to obj) Python objects, which is particularly useful for saving complex data structures like lists & dictionaries

⇒ WRITING

-
```
import pickle
data = {"name": "Alice", "age": 78}
with open ("svt.pkl", "wb") as f:
pickle.dump (data, f)
```

⇒ READING

-
```
import pickle
with open ("svt.pkl", "rb") as f:
data = pickle.load (f)
print (data)
```

CSV FILES

Reading and writing in CSV Files

⇒ WRITING

```
import csv
data = [["Name", "Age"],
["Alice", 18], ["Jake", 21], ["Jew", 20]]
with open("sut.csv", "w", newline="") as f:
writer = csv.writer(f)
writer.writerows(data)
```

⇒ READING

```
import csv
reader = csv.reader(f)
for row in reader:
    print(row)
```

Exceptions and Error Handling

- crucial for managing errors that may occur during file operations, such as file not found errors or permission errors.

EXCEPTION : contradictory or unexpected situation/ error, during program handling, is known as exception.

ERROR HANDLING: way of handling anamalous situations in a program - run, is known as exception /error handling.

→ Division by zero error
→ Access the elements of an array beyond range.
→ Invalid input.
→ Hard disk crash → Heap memory crash
→ opening a non existing form

```
try:
    with open ("cfg.txt", "r") as f:
    data = f.read()
except FileNotFoundError:
    print ("The file doesn't exist")
except Exception as e:
    print (f"An error occured : {e}")
```

Regular Expression Operations

Using special characters in Regular Expressions
- special characters allow for complex matching patterns

. : matches any character except a newline.
^ : matches the start of a string
$: matches the end of a string
* : matches 0 or more repititions of the preceding character.
+ : matches 1 or more repititions of the preceding "
? : matches 0 or 1 repition " " "
\d : matches any digits (equivalent to [0-9])
\w : matches any alphanumeric char ([a-z, A-Z, 0-9])
\s : matches any whitespace char (space, tab, newline)
[] : matches any single char. inside brackets
| : ~~nn~~ acts as an OR operator
{m,n} : matches from m to n repititions of the proceding character.

Regular Expression Method
- Python's re module provides various methods for working with regex.

```
import re

text = "The price is $20 for 2 items."

# Search for pattern
match = re.search(r'\$\d+', text)
if match:
    print("Found:", match.group())
```

Output : $20

```
# Find all matches
matches = re.findall (r'\d+', text)
print ("All matches :", matches)
```
output: ['20', '2']
```
# Replaces with sub
new_text = re.sub (r'\d+', 'X', text)
print ("Replaced text:", parts)
```
output: ['The', 'price', 'is', '$20', 'for', '2', 'items']

Named Groups in Python Regular Expressions

```
pattern = r"(?P<currency>\$)(?P<amount>\d+)"
text = "The price is $20"

match = re.search (pattern, text)
if match:
        print ("currency:",
match.group ("currency"))
```
output: $
```
        print ("Amount:",
match.group ("amount"))
```
output: 20

Regular Expression with glob Module

glob module: used to find files & directories based on a pattern. It is often used for file matching

```
import glob
txt_files = glob.glob ("*.txt")
print ("Text files :", txt_files)
```
output: (Lists all .txt files in the current directory)

Python Packages

- python has a vast ecosystems of packages that simplify complex operations.

1. Matplotlib (used for data visualization)

```
import matplotlib.pyplot as plt

x = [1,2,3,4]
y = [10,15,13,18]

plt.plot(x,y, marker='o')
plt.title("Line plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

2. Numpy (provides support for large, multi dimensional arrays & matrices, along with math functions)

```
import numpy as np

arr = np.array([1,2,3,4])
print("Array:", arr)

print("Mean:", np.mean(arr))
print("sum:", np.sum(arr))
```

3. Pandas (data manipulatn and analysis, offering data structurs like series and DataFrame)

```
import pandas as pd

data = {'Name': ["Alice", "Jake", "Christy"],
        'Age': [20, 23, 25]}
df = pd.dataFrame(data)

print("Data:\n", df)
print("Describe:", df.describe())
print("Filter:", df[df['Age'] > 23])
```

4. Keras (for building & training deep ~~tevet~~ learning models)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

#Simple neural network
model = Sequential([
        Dense(64, activation='relu',
input_shape=(10,)),
        Dense(1, activation='sigmoid')
])

#compile the model
model.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])
print(model.summary())
```