🚀 21-Day Python Challenge

Day 19 – Mini Projects

GK InfoTech Solutions – Institute of Engineers

🎯 Learning Objectives

By the end of today's class, you will be able to:

- Combine all the Python concepts learned so far.
- Build real-world console applications.
- Improve logical thinking and problem-solving skills.
- Learn how professional programmers divide a problem into smaller tasks.

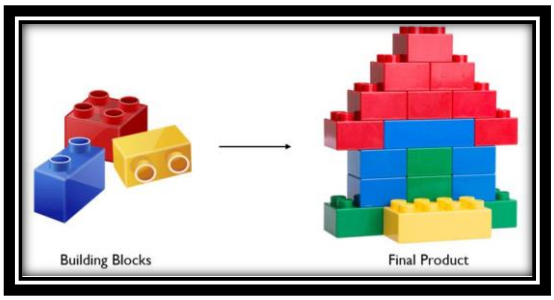
📝 Why Mini Projects?

Until now, you have learned:

- Variables
- Data Types
- Conditions
- Loops
- Functions
- Lists
- Dictionaries
- File Handling
- OOP
- Exception Handling

Now it's time to **combine all these topics into complete programs.**

📷 Picture 1 – Building a House



Explanation

```
Bricks
   ↓
Walls
   ↓
Rooms
   ↓
House
```

Programming is similar:

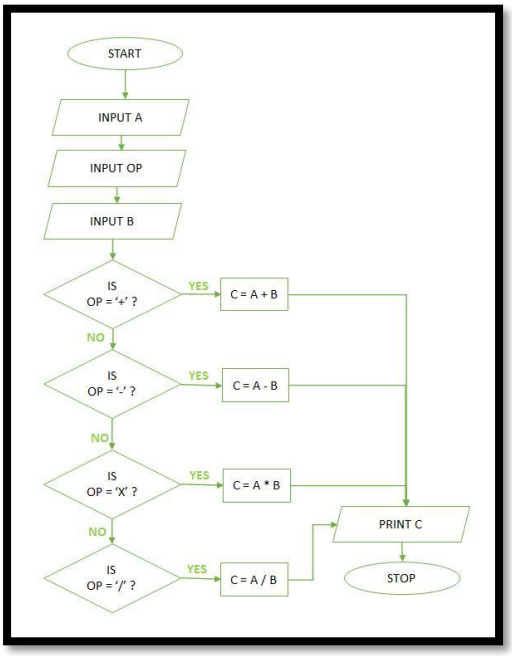
```
Variables
   ↓
Conditions
   ↓
Loops
   ↓
Functions
   ↓
Complete Project
```

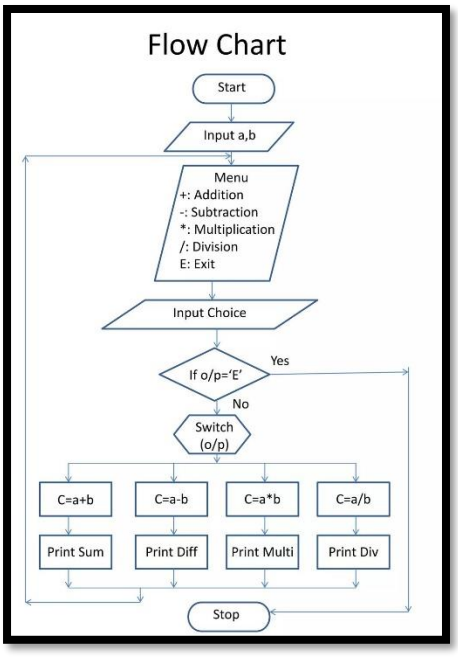
🧮 Mini Project 1 – Calculator

What Will We Learn?

- User Input
- Conditions
- Arithmetic Operators
- Loops

📷 Picture 2 – Calculator Flow





Start

↓

Enter Numbers

↓

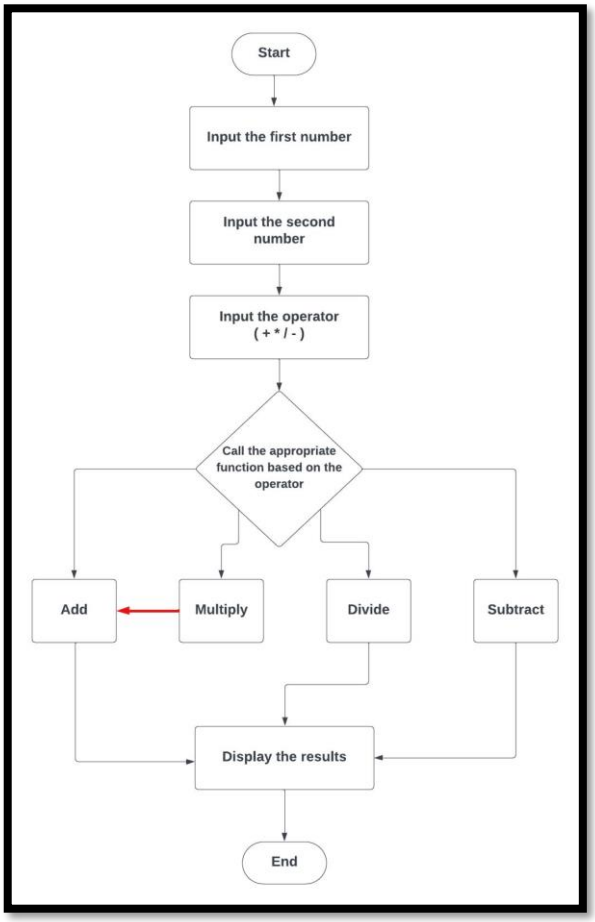
Choose Operation

↓

Calculate

↓

Display Answer



Program

```
print("===== SIMPLE CALCULATOR =====")

while True:

    print("\n1.Add")
    print("2.Subtract")
    print("3.Multiply")
    print("4.Divide")
    print("5.Exit")

    choice = input("Enter Choice : ")

    if choice == "5":
        print("Calculator Closed")
        break

    num1 = float(input("Enter First Number : "))
    num2 = float(input("Enter Second Number : "))

    if choice == "1":
        print("Answer =", num1 + num2)

    elif choice == "2":
        print("Answer =", num1 - num2)

    elif choice == "3":
        print("Answer =", num1 * num2)

    elif choice == "4":

        if num2 == 0:
            print("Cannot Divide by Zero")
        else:
            print("Answer =", num1 / num2)

    else:
        print("Invalid Choice")
```

Sample Output

```
===== SIMPLE CALCULATOR =====

1.Add
2.Subtract
3.Multiply
4.Divide
5.Exit

Enter Choice : 1

Enter First Number : 25
Enter Second Number : 15

Answer = 40
```

🎮 Mini Project 2 – Number Guessing Game

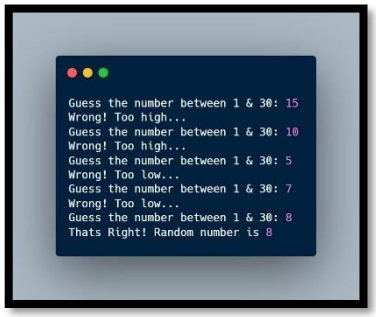
Objective

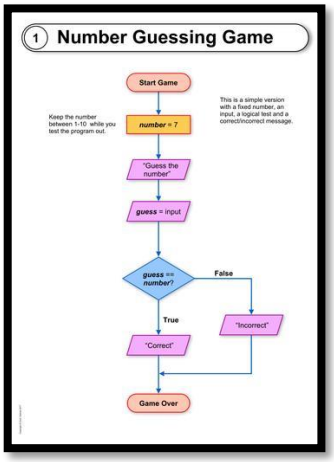
The computer chooses a random number.

The player guesses it.

If the guess is wrong, the computer gives hints.

📷 Picture 3 – Guess the Number





Program

```
import random

secret = random.randint(1,10)

while True:

    guess = int(input("Guess Number (1-10): "))

    if guess == secret:
        print("Congratulations! You Won!")
        break

    elif guess < secret:
        print("Too Small")

    else:
        print("Too Large")
```

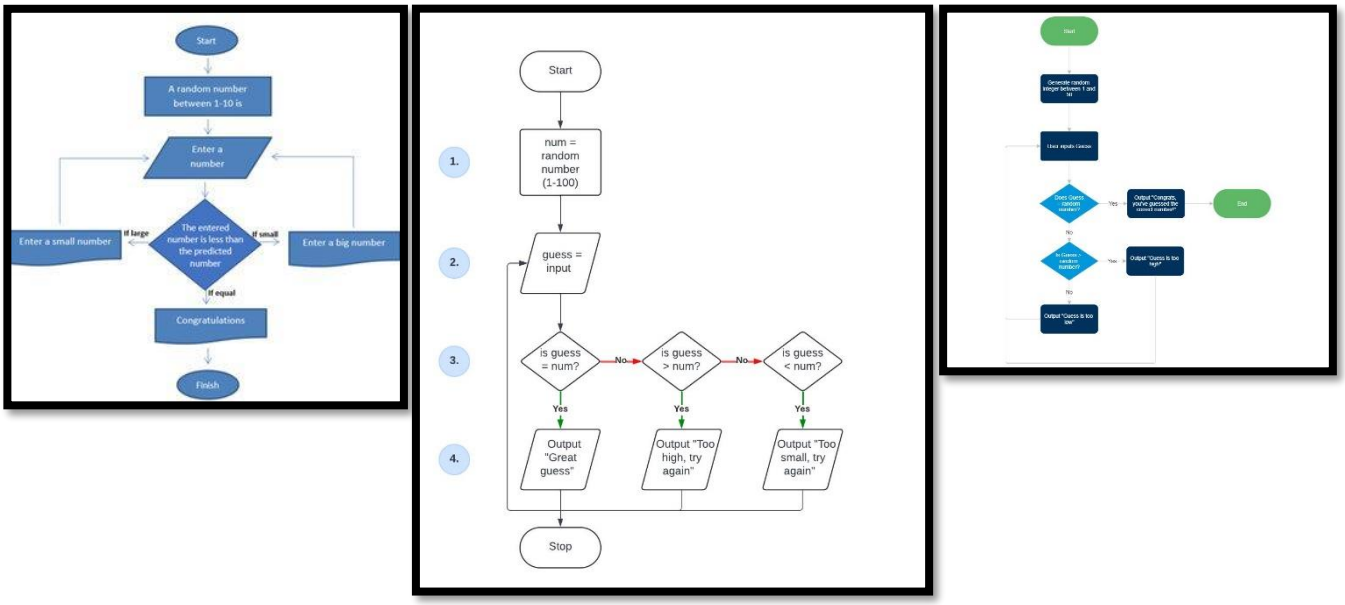
Sample Output

```
Guess Number (1-10): 3

Too Small

Guess Number (1-10): 8

Too Large

Guess Number (1-10): 6

Congratulations! You Won!
```

🧠 How the Game Works

```
Computer

↓

Random Number

↓

Player Guess

↓

Correct?

↓

Yes → Win

↓

No

↓

Give Hint

↓

Guess Again
```

📷 Picture 4 – Guessing Game Flowchart



Assignment

Build:

- Student Result Calculator
- ATM Machine
- Library Management (Simple)
- Quiz Game
- Multiplication Table Generator

🌟 Day 19 Success Message

"Projects are where learning becomes experience. Every project you complete increases your confidence and prepares you for real software development."

🚀 21-Day Python Challenge

Day 20 – Intermediate Python

GK InfoTech Solutions – Institute of Engineers

🎯 Learning Objectives

Students will learn:

- Iterators
- Generators
- Useful Built-in Functions

These concepts help write cleaner, faster, and more memory-efficient Python programs.

📓 What is an Iterator?

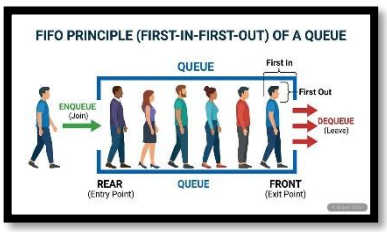
An **Iterator** is an object that lets us access items **one by one** instead of all at once.

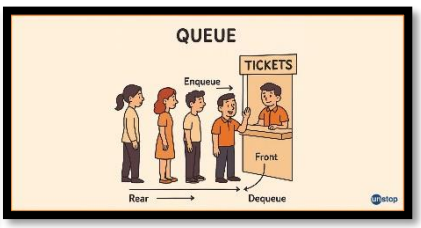
Real-Life Example

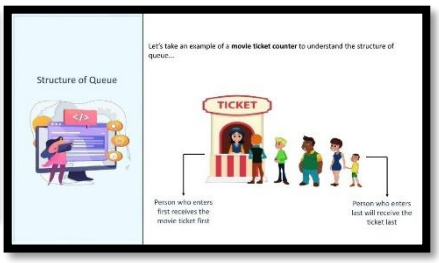
Think of a queue at a ticket counter.

People are served one after another.

📷 Picture 5 – Iterator







Explanation

```
List

↓

Rahul

↓

Anita

↓

Ravi

↓

Priya

↓

One Item at a Time
```

Example

```
students = ["Rahul", "Anita", "Ravi"]

it = iter(students)

print(next(it))
print(next(it))
print(next(it))
```

Output

```
Rahul
Anita
Ravi
```

How It Works

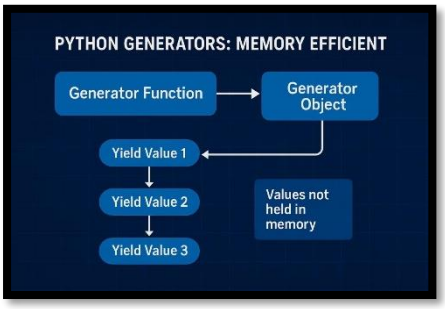
```
List

↓

Iterator

↓

next()

↓

One Value
```

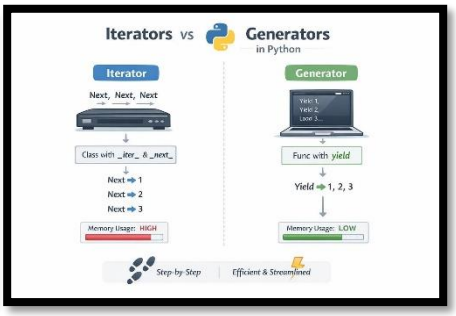
What is a Generator?

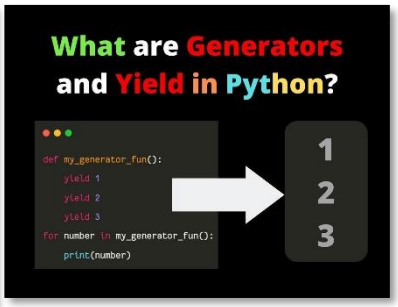
A **Generator** is a special function that returns values **one at a time** using the `yield` keyword.

Unlike `return`, which ends the function, `yield` pauses it and continues later.

Picture 6 – Generator







Water Tank Analogy

```
Water Tank
↓
Tap Opens
↓
One Glass
↓
Another Glass
↓
Another Glass
```

A generator produces values only when needed.

Example

```
def numbers():

    yield 1
    yield 2
    yield 3

for n in numbers():
    print(n)
```

Output

```
1
2
3
```

Difference Between `return` and `yield`

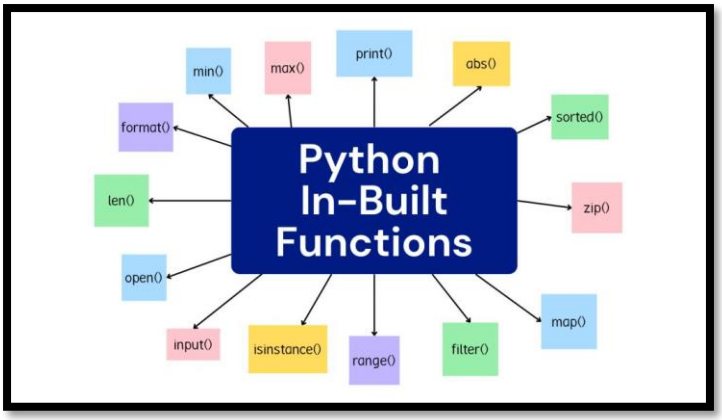
return	yield
Ends the function	Pauses the function
Returns one value	Produces multiple values one at a time
Uses more memory for large data	Saves memory by generating values as needed

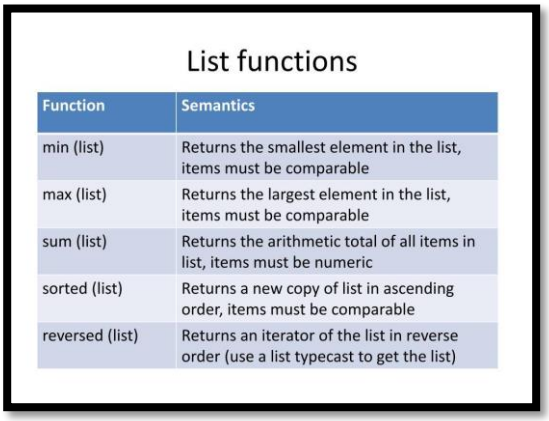
---

📓 Useful Built-in Functions

Python includes many built-in functions that make programming easier.

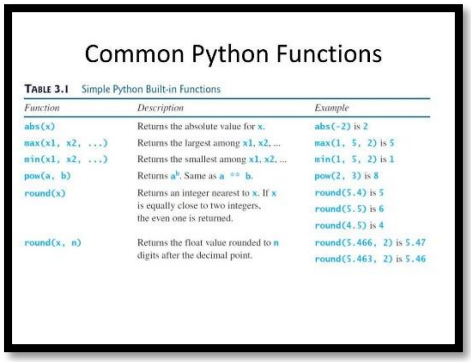
📷 Picture 7 – Python Built-in Functions





List functions

Function	Semantics
min (list)	Returns the smallest element in the list, items must be comparable
max (list)	Returns the largest element in the list, items must be comparable
sum (list)	Returns the arithmetic total of all items in list, items must be numeric
sorted (list)	Returns a new copy of list in ascending order, items must be comparable
reversed (list)	Returns an iterator of the list in reverse order (use a list typecast to get the list)



Common Python Functions

TABLE 3.1 Simple Python Built-in Functions

Function	Description	Example
abs(x)	Returns the absolute value for x.	abs(-2) is 2
max(x1, x2, ...)	Returns the largest among x1, x2, ...	max(1, 5, 2) is 5
min(x1, x2, ...)	Returns the smallest among x1, x2, ...	min(1, 5, 2) is 1
pow(a, b)	Returns a^b. Same as a ** b.	pow(2, 3) is 8
round(x)	Returns an integer nearest to x. If x is equally close to two integers, the even one is returned.	round(5.4) is 5 round(5.5) is 6 round(4.5) is 4
round(x, n)	Returns the float value rounded to n digits after the decimal point.	round(5.466, 2) is 5.47 round(5.463, 2) is 5.46

1. `len()`

```
names = ["Rahul","Anita","Ravi"]

print(len(names))
```

Output

```
3
```

2. `max()`

```
marks = [45,67,90,78]

print(max(marks))
```

Output

```
90
```

3. `min()`

```
marks = [45,67,90,78]

print(min(marks))
```

Output

```
45
```

4. `sum()`

```
marks = [45,67,90]

print(sum(marks))
```

Output

```
202
```

5. `sorted()`

```
numbers = [8,2,5,1]

print(sorted(numbers))
```

Output

```
[1,2,5,8]
```

6. `type()`

```
age = 20

print(type(age))
```

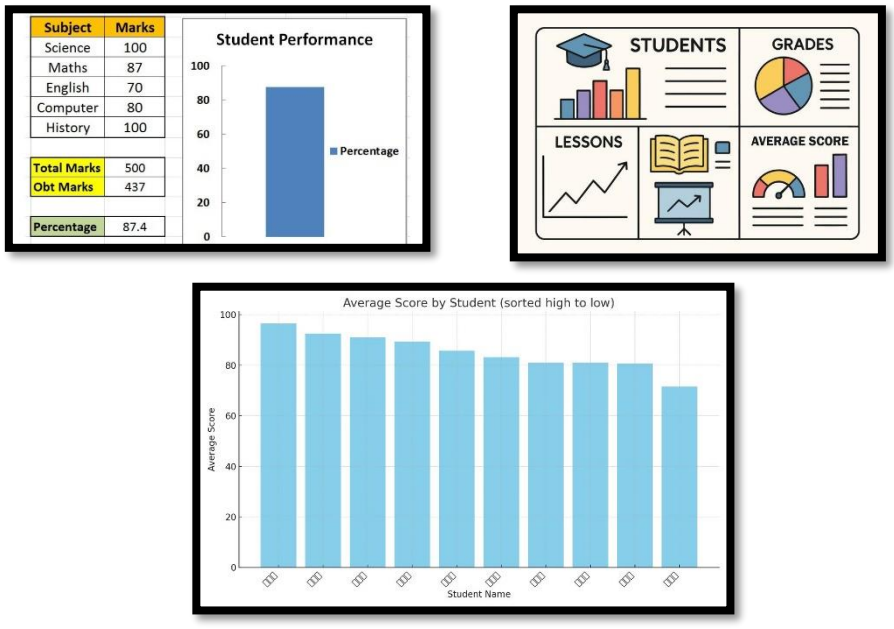
Output

```
<class 'int'>
```

Real-Time Example

```
marks = [55, 72, 81, 46, 90]

print("Total Marks :", sum(marks))
print("Highest Mark :", max(marks))
print("Lowest Mark :", min(marks))
print("Number of Students :", len(marks))
print("Sorted Marks :", sorted(marks))
```

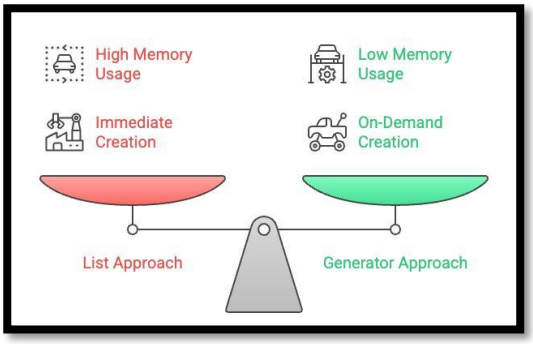
📷 Picture 8 – Student Marks Analysis

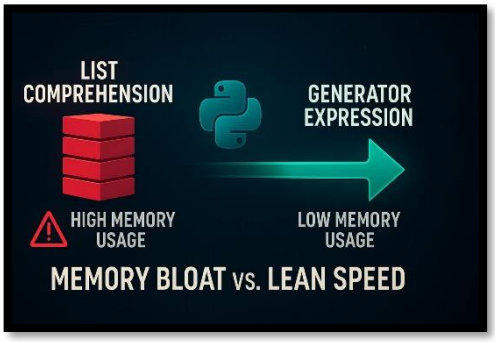


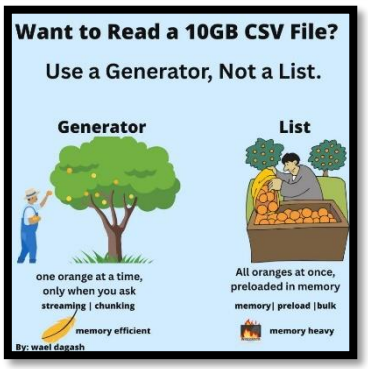
Memory Comparison

```
List

↓

Stores All Values

↓

Uses More Memory

------------------------

Generator

↓

Creates Values

↓

Only When Needed

↓

Uses Less Memory
```

📷 Picture 9 – List vs Generator







Assignment

Write programs to:

1. Create an iterator from a list.
2. Use `next()` to print values.
3. Create a generator using `yield`.
4. Find the largest number in a list.
5. Find the smallest number.
6. Find the total using `sum()`.
7. Sort a list using `sorted()`.
8. Find the data type using `type()`.

Interview Questions

1. What is an Iterator?
2. What is the purpose of `iter()`?
3. What does `next()` do?
4. What is a Generator?
5. What is the difference between `return` and `yield`?
6. Name five useful built-in functions in Python.
7. Why are generators memory-efficient?

✅ Day 20 Success Message

Congratulations! 🎉

Today you learned advanced concepts used by professional Python developers:

- ✅ Iterators
- ✅ Generators
- ✅ Useful Built-in Functions

These features help you write programs that are **efficient, clean, and scalable**.

"Programming is not only about making code work—it is about making code simple, efficient, and easy to maintain. Mastering iterators, generators, and built-in functions is an important step toward becoming a confident Python developer."