Lecture 02

Agenda

- For loops
- While loops
- Lists Functions
- Random module

Loops

- Loops are essential programming constructs that allow us to repeat a block of code multiple times. They are crucial for automating repetitive tasks and iterating through collections of data.
- **For** loops:

```
for i in range(0,1):
        print(i)
```

- **While** loops:

```
count = 0
while count < 5:
   print(count)
   count += 1
```

Lists

- In Python, a list is a data structure used to store an ordered collection of items. Lists are one of the most versatile and commonly used data types in Python. Here's how you can work with lists:
- **Creating a List:** You can create a list by enclosing a comma-separated sequence of items within square brackets [].
  my_list = [1, 2, 3, 4, 5]
- **Accessing Elements:** You can access individual elements of a list using indexing. Python uses 0-based indexing, so the first element is at index 0.
  first_element = my_list[0] # Retrieves the first element (1)
- **Modifying Lists:** Lists are mutable, which means you can change their elements.
  my_list[2] = 42 # Changes the third element to 42

Lists (continued)

- **Adding Elements:** You can add elements to the end of a list using the append() method. **Example:** my_list.append(6) # Adds 6 to the end of the list
- **Inserting Elements:** You can insert elements at a specific position using the insert() method. **Example:** my_list.insert(2, 99) # Inserts 99 at index 2
- **Removing Elements:** You can remove elements by value using the remove() method or by index using the pop() method. **Example:**
  my_list.remove(4) # Removes the element with the value 4
  popped_element = my_list.pop(1) # Removes and returns the element at index 1
- **List Length:** You can find the number of elements in a list using the len() function.
  length = len(my_list) # Returns the number of elements in the list
- **List Concatenation:** You can concatenate two or more lists using the ‘+’ operator
  combined_list = my_list + [7, 8, 9]

Function

- **Functions in Python** are reusable blocks of code that perform specific tasks. They allow you to encapsulate logic, making your code more organized, readable, and maintainable.
- **Defining a Function**
  In Python, you define a function using the **def** keyword, followed by the function name and parameters, if any.
- **Function Parameters and Return Values**
  Functions can accept parameters (input) and return values (output). Parameters allow you to pass data into the function, and return values allow the function to provide a result.

  Example:

```
def add(x, y): #defining a function
        result = x + y
        return result
sum_result = add(3, 5) #Calling a function
```

Practice Problems

1. Using For loop, given a range of numbers, only print the even numbers
2. FizzBuzz
3. Coin toss, Dice roll
4. Who’s going to pay?
5. Rock, Papers, Scissors
6. Highest, average and minimum marks scored from a list of marks
7. Password Generator
8. Calculator using functions

Fizzbuzz problem statement

Write a program that prints the numbers from 1 to **N**. But for multiples of 3, print "Fizz" instead of the number, and for the multiples of 5, print "Buzz" instead of the number. For numbers that are multiples of both 3 and 5, print "FizzBuzz".

Rules:

1. Print "Fizz" for multiples of 3.
2. Print "Buzz" for multiples of 5.
3. Print "FizzBuzz" for multiples of both 3 and 5.
4. For all other numbers, print the number itself.

Output Example:

```
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

Rock, papers, scissors

1. Take input from user (0, 1, 2), where 0 is rock, 1 is paper, 3 is scissors
2. Generate random input from computer (0-2), same as user
3. Compute using conditional statements to find out who won
4. Print winner

Example:

User Input = 0
Computer = 0

Output: “Its a draw”