

# **CORE PYTHON**

## **PROGRAMMING AND PROBLEM ANALYSIS**

Kazi Sakib Hasan  
BRAC University, Department of CSE

## Brief Contents

Preface	03
<a href="#">Chapter 01 Core Concepts</a>	06
1.1 Python Program	07
1.2 Variables	08
1.2.1 Variable naming conventions and rules	10
1.3 Operators	11
1.4 Data Structures, Algorithm & Data Types	19
1.5 Type Conversion	21
1.6 Indentations & Comments	24
1.7 Keywords & Built-in Modules	26
1.7.1 Keywords	26
1.7.2 Built-in Modules	27
1.7.3 Input prompt and format() function	32
1.8 Conditional Statements	35
1.9 Operator Precedence	42
Exercises	
<a href="#">Chapter 02 Loops</a>	50
2.1 For loop	50
2.2 While loop	56
2.3 Infinite loop	59
2.4 Early Exit	61
2.4.1 Implementations of loop terminators in while loop	62
2.4.2 Implementations of loop terminators in for loop	63
2.5 Program Analysis	64
Exercises	79
<a href="#">Chapter 03 Sequence Data Types</a>	87
3.1 String	87
3.1.1 String Attributes	87

3.2.2 Program Analysis	100
String Exercises	107
3.2 List	114
3.2.1 List Attributes	115
3.2.2 Program Analysis	123
List Exercises	131
3.3 Tuples	138
3.3.1 Tuple Attributes	138
3.3.2 Program Analysis	141
<a href="#">Chapter 04 Dictionary</a>	144
4.1 Dictionary Attributes	144
4.2 Program Analysis	149
Exercises	156
<a href="#">Chapter 05 User Defined Function</a>	162
5.1 Structures of a User Defined Function	164
<a href="#">Chapter 06 Hand Simulation</a>	171
Exercises	178
Tracing	178
Debugging	184
<a href="#">Appendices</a>	
Appendix 1: Unary Operations	186
Appendix 2: Built-in Functions	187
Appendix 3: Differences between the Data Structures	191
Appendix 4: Sorting Algorithms	193
Appendix 5: Boolean Operations	196
Glossary	197
Advanced Practice Problems	198
References	233

# Preface

The book is written for the purpose of developing the critical thinking ability and programming skills of new Python learners. I started writing the book when I was doing the CSE110 course of BRAC University in my undergraduate second semester. But the interesting thing is, I did the same course in my first semester as well. So, yes, in my second semester I did retake the course because I got a worse result (1.3) there. However, I became more attentive to the course by taking notes and spending more and more time on problem solving, which later brought a severe positive transformation of me in the course. That semester, I ended up attaining 4.00 in CSE110. While I was taking notes, I kept compiling and saving everything in a google doc file that eventually enlightened me with the idea of turning the notes into a book as there is no existing text book on that course and hence the students struggle a lot to understand the concepts. A significant number of students in the first semester, CSE department are put on academic probation due to a very bad result on that course (so do I). As a result, I deeply felt the intricacies of the probationary days and so I set up my mind to help the students. Before my second semester began, I started analyzing the course materials profoundly and started to solve the assignment problems that I kept collecting from the first semester. In a few weeks, I mastered the typical problems. Then the book writing process was initialized. I compiled the class notes, added typical problems, then created advanced problems by myself. After the structure was ready, I sent the draft version to my previous CSE110 instructor and a senior lecturer of the university, not willing to say the name. He praised the work and assured me of a collaboration. Unfortunately, he cut off the connection with me and hinted at discarding the project. Nevertheless, I decided to upload the soft copy of the book so that the students do not need to wait a long time in uncertainty.

**Scope :** This book offers the students the following advancements :

1. Detailed information about each concept.
2. Related examples to understand each topic.
3. Problem analysis (algorithm visualization) for deep understanding.
4. Adequate practice problems and advanced practice problems, debugging problems and tracing problems.
5. I wrote the book in the easiest way possible. The usage of difficult words and grammar is very less.

**Disclaimer :** This book is written by an undergraduate student and not yet reviewed by any professionals on the field. Therefore, certain writing styles might contain unprofessionalism and minor editing mistakes might exist too. In case you want to send me feedback about the book, kindly email me here: [simanto.alt@gmail.com](mailto:simanto.alt@gmail.com)

**Acknowledgements** : This book is written based on the CSE110 course materials provided by buX (Brac University online learning platform), class lectures of Ms. Mahjabeen Tamanna Abed (Lecturer, Brac University) and Md. Saiful Islam (Senior Lecturer, Brac University).



# Chapter 01: Core Concepts

Before getting into the main theme, I would like to briefly discuss the conspectus of computer programming, predominantly about Python programming. The primary purpose of the computer was to “compute” mathematical expressions when it was first built. Day-by-day the base functions of computers enhanced and hence the field and work range of computers increased. A computer needs a command to accomplish tasks. Computer programming is the only way to command and communicate with a computer which is done by following a specific programming paradigm. A **programming paradigm** is a style that shows how certain source codes will be executed. Simple meaning: a programming paradigm is an approach to solve a specific problem. A problem can be solved in different ways, every paradigm has its identical characteristics that finishes a task in a specific method. Python supports three types of programming paradigms including **structured programming** (usually imperative and procedural programming), **object-oriented programming (OOP)** and **functional programming paradigms**. Most of the program written and discussed in this book is **imperative programming**, more precisely, **procedural programming**, a paradigm that uses commands and statements to carry out operations. The programming paradigms can be separated into two major paradigms, imperative and declarative. Imperative program visualizes how a program works step-by-step, we can simply understand the way a program works by learning the imperative paradigm of programming. Usage of functions is not really necessary in imperative programming. On the other hand, a declarative program focuses more on directly solving a problem instead of understanding the problem solving approaches and algorithms. A computer programmer writes a script to design an executable computer program, which can complete a specific objective and the program executes abide by the control flow and returns the output to the programmer. Meanwhile, object-oriented programming (OOP) processes everything converting those into objects and classes. [1]

On the basis of the need of the programmer, a computer program can be about anything random. Maybe he can write scripts for a program that can sum as many numbers as he wants, or maybe a difficult equation like the formula of the moment of inertia of a flywheel about its axis of rotation which is too tough to evaluate, he can write scripts to measure heat from given values of  $m, s$ , temperature difference and even he can develop a program that can tell him

about his friend's mental status based on some data that he would input about him. So, if the programmer has critical thinking ability and creativity, he may use computer programming in every sphere of his life including academic and non-academic activities. The entire process of designing a computer program is known as “coding” or “programming.”

**About Python:** Python is a general-purpose programming language (GPL), a language that can be used to develop a variety of programs and it is not confined to accomplish some specific tasks only. Python is often used to build websites and software, task automation, data analysis, data visualization, implement artificial intelligence and most specially; complete tasks of our daily life. Python was created by a Dutch computer programmer named Guido Van Rossum and first released on 20th February, 1991. [2]

In this chapter, we will try to understand the concepts of data structures, algorithm, variables, operators, indentations, keywords, conditional statements, data types (that include text, numeric, sequence, mapping and boolean type), built-in functions like `print()` and `input()`. Well obviously, we will discuss every topic rigorously when it's needed. Note that, we will be using Python 3 to execute our codes. Suggested Integrated Development Environment (IDE): Google Colaboratory, as no download and installation process is required there. Just log in, open a new notebook and write your script. You can use Spyder, Anaconda, PyCharm, VS Code or other IDE's too if you want. But some functions might feel different from Google Colab.

## 1.1 Python Program

A python program is developed by a bunch of statements, also known as commands which are scripted by an user. The programs are written on an IDE that has its idiosyncratic interface consisting of an **interpreter** : which executes the codes, **cells/shells**: on which the codes are written, **console**: where the output is shown to the programmer. Python programs work with objects, these objects have a type (int, float, str etc.) of their own that identify which type of operation can be operated on the objects. A sample program is shown below. This is the traditional program which is programmed by most of the computer science and programming newcomers as their first program. “Hello world” was first written by a Canadian computer scientist named Brian Kernighan in 1972. [3] Overnight, this became a trend. A lot of computer science teachers and students use the short line as their first written program.



```
>>> print("Hello world")
Hello world
```

The program shown above is scripted by a programmer, written on a cell of an IDE, executed by an interpreter and printed on the output console. This code is executed and shown to the user following 2 steps. Firstly, the codes are compiled into **byte codes**. Then, the **byte codes** are converted into **machine codes** by a Python interpreter as computers can only understand machine code. Finally, the machine code instructions are executed by the computer's processor and outputs are displayed.

A little off topic about the interface of the book: In this book, I followed the interface of Google colab and Python 3.8 both for representing a cell and the output console. In the raw Python 3.8, ">>>" determines the line where codes are written, "..." denotes the indentations for conditional statements and loops. Output is printed in a fresh new line after the codes are done scripted. Google colab looks different. The codes are written without any >>> sign and spaces before each line indicate the indentations. Outputs are displayed in a new line.

The interpreter of Python processes the code from **left to right, beginning to the end**. The execution and output of the codes is done and shown according to the exact manner as well.

```
>>> print("Hey there.")
>>> print("I'm Stealth.")
>>> print("And she is my friend Etherea.")
Hey there.
I'm Stealth.
And she is my friend Etherea.
```

## 1.2 Variables

Variables are the “thing” that stores a random data type. Precisely, variables work as a container that contains a set of bits or types of data that are known as “*values*”. Generally, each variable stores only a specific type of data (until you’re making a list/tuple/dictionary to store inside a variable. Lists, tuples and dictionaries are multi-element data structures that can contain several types of data.) Variables are auto created when you assign a value to it. For example, imagine

adding two integer numbers, let's suppose 7 and 3 (every positive and negative rational number excluding fractional numbers belong to the set of integer numbers). So, if we want to sum two integer numbers then we'll simply write it and press ctrl+enter or click the run button instead (for google colab). It'll print/show us the result. But the result isn't stored anywhere. It's like that the shown output is drowned in a sea of memory from where it is shown to us and we can't retrieve it anyhow. Hence, we can't "use" that output result again. If we want to add another integer number with the output result then it's impossible, as the result isn't stored anywhere. So it's necessary to store the value first. There comes the variables to solve this issue. Variables create a storage location to reference the stored value that can be accessed anytime. Thus, it's possible to retrieve a stored value and use that over and over again. Now let's create two variables and store the integers inside it. We can name the variables num1 and num2.

```
>>> num1 = 7
>>> num2 = 3
>>> sum = num1+num2
>>> print(sum)
10
```

Look at the third line. There we created a variable named 'sum' to store the summed value of the two variables num1 and num2 which respectively contain two integer type data 7 and 3. Remember that variables must be created at the beginning of a line as the assignment process is left to right. We can't write 3=var1 to denote that the variable "var1" is storing 3. Rather it means the variable is 3 and the value is var1. Since a variable name can't be 3 so the program actually has no meaning and it would show us an error message in the console.

A random value of a variable is replaced if a new value is assigned to it. Suppose a variable named var1 that stores an integer value 2. If we again assign a new value to the same variable, then the previous value won't be counted. This phenomenon is known as *scoping*.

```
>>> var1 = 2
>>> var1 = 4
>>> print(var1)
4
```

This happens because Python interprets source code from first line to last line, left to right.

## 1.2.1 Variable naming conventions and rules

Do we know the definitions and differences between rules and conventions?

**Rules:** The obligations that someone must follow.

**Conventions:** The suggestions one is inspired to follow but not necessary.

While creating variables, we must follow the rules (or else we'll get errors!) of creating variables and we might follow some conventions as well. Variable naming rules are mentioned here:

1. We cannot start with numerical characters. For example: 1number, 2Hash, 3\_boom
  2. We cannot start with special characters like @, #, \$ etc.
  3. We can only name our variables with alphabets, that can be either in uppercase or lowercase. It's OK to begin with an underscore as well.
  4. Variables' name can't be separated by a "space." For example: Number one, number two, a word etc. Rather we can name those like Numberone, numbertwo, aword.
  5. Variable names are case-sensitive. "Real" and "real" won't represent and work as the same variable.
  6. Python keywords can't be used as variable names. There are like 33 keywords in Python
3. We will discuss them soon.

### Variable naming conventions:

Two significant conventions are mentioned below.

1. Check out the 4th point of variable naming rules. Doesn't it look so weird to see the names like Numberone, numbertwo, aword and these are tough to understand as well? So we can do a simple thing to make it more understandable and better. We can separate the words with an underscore (\_). For instance, Number\_one, number\_two, a\_word; these names sound better and are more understandable. Separating words with an underscore to make an easy to understand word is known as snake casing.
2. We better not name our variables with only one alphabet like a, b, c, d etc. When writing short programs maybe it won't hamper our work that much but in case of larger programs we may get confused to understand which variable contains which items. It's

good to practice creativity while naming our variables. We can name our variables in a proper way so that it can represent the stored data.

## 1.3 Operators

The word “Operator” came from the latin word “operator” which means work or labor. So etymologically operators are meant to carry out operations. In Python, operators operate the relations between variables and values. The thing on which the operator operates is known as an “operand.” Python operators can be divided into seven groups. They are: arithmetic operators, assignment operators, relational/comparison operators, logical operators, identity operators, membership operators and bitwise operators. Our discussion will be confined into arithmetic to membership operators.

**(a) Arithmetic operators** : These operators are used for tasks that are related to general mathematical calculations. Here are the 7 arithmetic operators :

+ (example:  $5+7 = 12$ )

- (example :  $5-7 = -2$ )

\* (example :  $5*7=35$ )

/ (example:  $9/3=3.0$ )

// (example:  $9//3 = 3$ ), (example:  $10//3=3$ )

% (example:  $4\%2 = 0$ ), (example:  $4\%3=1$ )

\*\* (example:  $4**2 = 16$ )

*NOTE* : The first three arithmetic operators are easy to understand. Just simple operators that operate general calculations. But the next four seem spiny.

*Divisional arithmetic operators “/” and “//”*

There’s actually a difference between divisions in Python. We can perform division in two ways. One is “division” and the other is “floor division”. “/” symbol represents the arithmetic

operator for division. The resultant (quotient) that shows in the console will be a float data type if we use “/” to divide something by something. Shown in the example, though the resulting value hasn’t any remainder, the output will still simply put a “.0” next to the quotient to make it a float type data. In contrast, if we use the “//” (floor division) operator then the resulting value will appear as an integer removing the fractional portion. Look at the example above,  $10//3 = 3$ . It didn’t show us 3.33.

### *Modulus arithmetic operator “%”*

This operator is also known as mod (%). It extracts the remainder of two divisible numbers. See the given example,  $4\%2$  (pronounced as four mod two) yielded 0. Because if we divide 4 by 2 then there actually no remainder remains. Hence it comes 0 as output.

### Clarification

Math:  $4 / 2$

Quotient : 2

Remainder : 0

So,  $4\%2 = 0$

Similarly,  $4\%3$  is equal to 1 since 4 divided by 3 yields “3” as quotient and “1” as remainder.

The universal formula of calculating remainder is,

Dividend = Divisor \* Quotient + Remainder

Assume a dividend 9 and divisor 2. According to the formula written above,

$9 = 2*4 + \text{Remainder}$

So, remainder =  $9-8 = 1$  (Ans)

There’s another way to evaluate the remainder using a calculator.

Step 1: Divide the dividend by the divisor.

Step 2: If the resultant is fractional, then extract the absolute value of the fractional part only.

Step 3: Multiply the fractional part with the divisor. The answer is the remainder.

Example 1: Find the remainder of 5/2 (Means  $5\%2=?$ )

Here, divisor = 2 and dividend = 5

$$\text{dividend/divisor} = 5/2 = 2.5$$

$$| \text{Fractional portion} | = 0.5$$

$$\text{Remainder} = | \text{Fractional portion} | * \text{divisor}$$

$$\text{So, remainder} = 0.5 * 2 = 1 \text{ (Ans)}$$

Example 2: Find the remainder of 7 divided by 3 (Means  $7\%3=?$ )

Here, divisor = 3 and dividend = 7

$$\text{dividend/divisor} = 7/3 = 2.33333$$

$$| \text{Fractional portion} | = 0.33$$

$$\text{Remainder} = | \text{Fractional portion} | * \text{divisor}$$

$$\text{So, remainder} = 0.33 * 3 = 1 \text{ (Ans)}$$

Example 3: Find the remainder of 5 divided by -2 (Means,  $5\%-2=?$ )

Here, divisor = -2 and dividend = 5

$$\text{dividend/divisor} = 5/-2 = -2.5$$

$$| \text{Fractional portion} | = 0.5$$

$$\text{Remainder} = | \text{Fractional portion} | * \text{divisor}$$

$$\text{So, remainder} = 0.5 * -2 = -1$$

Example 4: Find the remainder of -5 divided by 2 (Means,  $-5\%2=?$ )

Here, divisor = -2 and dividend = 5

$$\text{dividend/divisor} = -5/2 = -2.5$$

$$| \text{Fractional portion} | = 0.5$$

$$\text{Remainder} = | \text{Fractional portion} | * \text{divisor}$$

$$\text{So, remainder} = 0.5 * 2 = 1$$

Formula described above may not yield the correct result if the dividend is negative fractional. To resolve the issue we can use the following formula to evaluate remainder:

$$\text{Remainder} = x - y * (x//y).$$

*Example 1:  $-5.5 \% 2 = ?$*

Here,  $x = -5.5$  and  $y = 2$

$$\text{Remainder} = -5.5 - 2 * (-5.5//2)$$

$$= -5.5 - 2 * (-3)$$

$$= -5.5 + 6$$

$$= 0.5 \text{ (Ans)}$$

For larger values, even this formula may yield wrong results. Hence it's always better to use the **Dividend-Divisor-Quotient-Remainder** formula. The formula is,

$$\text{Dividend} = \text{Divisor} * \text{Quotient} + \text{Remainder}$$

While doing negative modulus, keep in mind that Python only returns **floored modulo**. Contrast modulo to this modulo is known as **truncated modulo**. Floored modulo and truncated modulo varies on the basis of the type of the divisor and dividend. If the divisor is negative and the dividend is positive then truncated modulo returns **positive** remainder and floored modulo returns **negative** remainder. Contrary to this one, if the divisor is positive but the dividend is negative, truncated modulo returns **negative** remainder and floored modulo returns **positive** remainder.

### **When the dividend is negative**

Suppose, we need to find out the result of  $-9\%4$ .

By using a calculator we get,  $-9/4 = -2.25$ . Removing the fractional part, the quotient is -2.

Applying the formula we get,

$$-9 = 4 * (-2) + \text{Remainder}$$

So, remainder = -1

It's clearly seen that the generalized Dividend-Divisor-Quotient-Remainder formula operates the **truncated modulo** but Python returns us the **floored modulo**. If we write `print(-9%4)` on our code editor we'll see the program will return 3, the floored modulo. It will not return -1 to us which is the truncated modulo.

To get the floored modulo, make sure that we take the nearest integer as the quotient, after doing the usual division in our calculator. By using a calculator we get  $-9/4 = -2.25$ . To get the truncated modulo we removed the fractional portion and took -2 as the quotient. But this time, we'll convert -2.25 to its nearest integer value which is -3 and then find out the **floored modulo**.

Now, applying the formula we get,

$$-9 = 4 * (-3) + \text{Remainder}$$

So, remainder = 3 (Ans)

### **When the dividend is positive**

I'm skipping the truncated modulo here. Since Python always returns the floored modulo.

Problem:  $9\%(-4) = ?$

Using calculator we get,

$$9/(-4) = -2.25$$



Nearest integer of -2.25 is -3. Therefore, quotient = -3

By applying the formula we get,

$$9 = (-4) * (-3) + \text{Remainder}$$

$$\text{Remainder} = -3 \text{ (Ans)}$$

**Note:** Open your code editor and write `print(5.5//2)` and run the program. The result you will see is 2.0. Now, write `print(-5.5//2)` and run it. This time, the resultant is -3.0. Why does this happen? Well, when we do a floor division the resultant is returned to us as the nearest integer value. So,  $5.5/2 = 2.75$ . Hence,  $5.5 // 2$  will be equal to 2. Note that, since the nearest integer is returned, the returned value is less than the resultant that was yielded when we operated normal division. The same thing occurs for the case when the dividend is negative.  $-5.5/2 = -2.75$ . Its nearest integer is -3.0 and -3.0 is less than -2.75.

If all of the methods stated above seem difficult to you then you may use this shortcut.

Assume we've to find the result of  $9\%-4$ .

$$\text{So, } 9/-4 = ?$$

Keep incrementing the  $|\text{divisor}|$  by  $+\text{divisor}$  until you exceed the  $|\text{dividend}|$ .

$$4+4 = 8$$

$$8+4 = 12$$

Now, subtract the  $|\text{dividend}|$  from the resultant. Put a negative sign before the resultant as the divisor is a negative number. This is the final answer.

$$12-9 = 3$$

$$-3 \text{ (Ans)}$$

If the dividend is negative, yet the process is the same. The difference is, we do not need to put a negative sign before the resultant. Means,  $-9/4 = 3$

*Exponential arithmetic operator (\*\*)*

The double asterisk (**\*\***) is used to denote the power. When we write mathematics we superscript an alphabet or number to determine powers (For example:  $2^2$ ,  $3^4$  and  $4^5$ ). In Python, we will write it like `2**2`, `3**4` and `4**5` respectively.

**(b) Assignment operators :** These operators operate to assign values to variables. Most common assignment operators are `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `//=` and `**=`. Look at the examples below to understand the concept thoroughly.

```
>>> a = 2
>>> print(a)    #Output: 2

>>> a += 1
>>> print(a)    #Output: 3
>>> a = a+1
>>> print(a)    #Output: 3
>>> #Here a+=1 and a=a+1 represent the same thing.

>>> #Example 2: Usage of -=
>>> a = 3
>>> a-= 2
>>> print(a)    #Output: 1
>>> #similar to previous:
>>> a = a - 2
>>> print(a)    #Output: 1

>>> #Example 3: usage of *=
>>> a = 5
>>> a*=2
>>> print(a)    #Output: 10
>>> #Because a*=2 means a = a*2. Therefore, 5*2 = 10

>>> #Example 4: Usage of /=
>>> a = 10
>>> a/=2
>>> print(a)    #Output: 5.0

>>> #Example 5: Usage of %=
>>> a = 10
>>> a%=2
```

```
>>> print(a)      #Output: 0
```

**Explanation:** Here,  $a\%=2$  means,  $a = a \% 2$ . Therefore, the result of `>>> 10%2` is the output.

```
>>> #Example 6: Usage of **=
>>> a = 4
>>> a**=2
>>> print(a)      #Output: 16
```

**(c) Relational/Comparison operators :** To show the comparison between two or multiple values relational operators are used. There are six comparison operators. They are `==` (equal), `!=` (not equal), `>` (greater than), `<` (less than), `>=` (greater than or equal), `<=` (less than or equal).

**(d) Logical operators:** Logical operators merge the conditional statements. The operators are “and”, “or”, “not”. Logical operators are usually collaboratively used with keywords like `if`, `else` and `elif`.

(i) Logical AND : The logical and operator returns True if both values/ boolean expressions are True. Otherwise, it returns False.

(ii) Logical OR: If only one value/boolean expression is True then this operator returns True.

(iii) Logical NOT: If the boolean expression is False then not operator returns True. For Trues, it returns False.

var1	var2	var1 and var2	var1 or var2	not var1
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Figure 1.1: The Truth Table of logical operators

**(e) Identify operators** : Identity operators are used to find analogies between the variables if they are the same thing or not. “is” and “is not” are the two identity operators.

Found a mistake? I didn't start writing the previous line with a capital letter. Right? Well, it's because “is” is an identify operator in Python but “Is” isn't!

**(f) Membership operators** : These operators check whether an item,word,letter,alphabet,sequence or any kind of object can be found in another object/variable or not. “in” and “not in” are the two membership operators.

**(g) Bitwise operators** : Bitwise operators are used to compare binary numbers. These operators are AND, OR, XOR, NOT, left shift and right shift.

## 1.4 Data Structures, Algorithm & Data Types

### Data structures

Data structures are the fundamentals of every programming language. Each kind of data structures has their own unique and common attributes that help a program to be built. Python has some built in data structures. Among them list, tuple, set & dictionaries are the most popular. These are the built-in data structures of Python.

### Algorithm

The step by step flowcharts of solving a problem can be defined as an “algorithm.” Let's get back into our first example where we summed up two integers. We can clarify the concept of algorithms using that example. Imagine creating an algorithm for adding two numbers. Each step will be a part of the algorithm.

*Step 1:* Create two variables and assign values to it using assignment operator (=).

*Step 2:* Create a third variable where the summation value will be stored.

*Step 3:* Using an arithmetic operator (+) carry out the summation process and assign the summed value to the third variable via “=” assignment operator.

*Step 4:* By using the `print()` built-in function show the output value on screen, the sum of two numbers.

Writing algorithms in a wrong order may terminate the code from execution, return unwanted or invalid outputs and also can cause program errors.

## Data Types

Python data types are separated into 4 groups. They are:

Numeric data type, sequence data type, mapping data type and boolean data type.

**(a) Numeric data type :** `int(integer)`, `float` and `complex`.

**Integer :** All rational numbers except fractional numbers belong to integer type data. The function for integer numbers is `int()`.

**Float :** The set of all decimal numbers are float type data. The function for float numbers is `float()`.

**Complex :** Mixtures of imaginary and real numbers are known as complex numbers.  $3 + 4i$  is a complex number, where 3 is a real number and  $4i$  is an imaginary number. Here  $i = \sqrt{-1}$ .

**(b) Sequence data type :** Sequence data types are used to store data inside a container and are accessible. List, tuple and string are the sequence data types. The functions for list, tuple and string are `list()`, `tuple()` and `str()` respectively.

**List :** List is a multi-element data structure that can store integers, strings or any kind of data types confining the data inside a `[]`. It's possible to store a list inside another list as well (nested list).

**Tuple :** Tuple is another kind of multi-element data structure similar to list but the elements inside it are immutable (unchangeable) unlike lists. Tuple stores values inside a `()` parenthesis.

**String :** String is a unique data type that is composed of varieties of alphabets, characters and numbers. Strings are assigned inside a `' '` or `" "`. Everything which is put inside those quotation marks are converted into strings.

**(c) Mapping data type :** Dictionary is the only one mapping data type in Python. It is made of a collection of keys and values where items are stored. The structure of a dictionary is {key:value}. Key and value may contain specific and different data types.

Tuple, string and dictionary will be discussed in detail in later chapters.

**(d) Boolean data type :** Boolean data type consists of two values: True and False. Based on written syntax and statements, the bool() function returns whether the statement is True or False. Example:

```
>>> bool(2<3)
>>> #Output: True

>>> bool(2>3)
>>> #Output: False
```

## 1.5 Type Conversion

It's possible to convert data types from one type to another. This is known as type conversion/type casting. There are some interruptions when we're type casting.

- Integer can only be converted to float and string.
- Float can be converted to string and integer.
- String can't be converted into integer or float until the assigned data type is numerical. Being numerical means the ASCII value of the characters are ranged between 48 to 57. The concept will be discussed in later chapters.

Passing an integer as an argument of the float() function will turn that integer into a float number. Integer to float conversion:

```
>>> user = 10
>>> print(type(user))
<class 'int'>

>>> user1 = float(user)
>>> print(user1)
>>> print(type(user1))
```

```
10.0
<class 'float'>
```

Note: Here the word 'class' defines the type.

Passing a float argument inside the int() function will turn that float into an integer number.

Float to integer conversion:

```
>>> user = 2.5
>>> print(int(user))
2
```

Integer and float to string conversion:

```
>>> user = 4
>>> user1 = 5.3
>>> print(str(user))
'4'
>>> print(str(user1))
'5.3'
```

Type casting shown above is done manually, by the programmer. This kind of type conversion is known as explicit type conversion. Python has two methods to convert the type.

**Implicit type conversion:** When the compiler changes a given data type, it is called implicit type conversion. If any of the operands are floating-point then the resultant becomes a floating-point too.

```
>>> print(4.0-2)
2.0
```

**Explicit type conversion:** When the user/programmer manually changes the data type of a certain data using the built-in function, it is called explicit type conversion. In this book, we'll mostly use these five built-in functions for explicit type casting.

1. str() : Returns a data type converting its type into a string.

```
>>> number = 10
>>> number1 = str(number)
```

```
>>> print(type(number))
>>> print(type(number1))
<class 'int'>
<class 'str'>
```

2. `float()` : Returns a numeric data type or the characters of ASCII range between 48-57 converting its type into a floating point number.

```
>>> string = '1123'
>>> integer = 22
>>> f1 = float(string)
>>> f2 = float(integer)
>>> print(type(string))
>>> print(type(integer))
>>> print(type(f1))
>>> print(type(f2))
<class 'str'>
<class 'int'>
<class 'float'>
<class 'float'>
```

3. `int()`: Returns a numeric data type or the characters of ASCII range between 48-57 converting its type into an integer.

```
>>> string = '12'
>>> floa_t = 12.34
>>> int1 = int(string)
>>> int2 = int(floa_t)
>>> print(type(int1))
>>> print(type(int2))
<class 'int'>
<class 'int'>
```

4. `tuple()`: Returns the data type converting it into a tuple. It can take various data types including list, string, dictionaries as the argument.

```
>>> number = '1'
>>> tupl1 = tuple(number)
```



```
>>> print(tupl1)
>>> print(type(tupl1))
('1',)
<class 'tuple'>
```

5. `list()`: Takes data as the argument and returns it a list. This function also receives different data types as the argument.

```
number = '2'
list1 = list(number)
print(list1)
print(type(list1))
```

## 1.6 Indentations & Comments

### Python indentations

Python indentations refer to spaces before a syntax. This feature is already used in this book. It has no readability advantage, rather Python indentations have an impact on the program. Improper indentations might cause errors and show undesired outputs. Python indentations also determine whether a certain block of code should be executed or not. When we'll be doing problems related to iterations, we will understand the importance of indentation more plainly. Yet a simple example is given below:

```
>>> var1 = 'String'
>>> if var1 == 'String' :
>>> print("True")
```

When you run the program, the console may appear with a message saying “IndentationError: expected an indented block”. But when you'll write the code in the following manner:

```
>>> var1 = 'String'
>>> if var1 == 'String':
...     print("True")
```

The program will return True.

## Python comments

Comments are simply written syntax begin with a “#”, which are usually written to understand a code, program, problem name, algorithm or anything related to or not related to a program.

We can write whatever we want as comments, it's up to our sweet will. Comments aren't executed and have no influence on a program. Using a comment feature is totally optional for a user. To write a comment, type # and then write anything you want. If you want to cancel a code line/syntax, then instead of clearing it with backspace you can put # before the syntax instead. The whole line will be turned into a comment and won't be executed. This is helpful when you're dealing with larger codes and tracking the values of multiple variables. You do not need to erase a line, just put a # before the syntax and it won't be executed.

Here's an example shown below.

```
#Following program subtracts two specific float numbers.
```

```
num1 = 2.5
num2 = 1.0
sub = num1 - num2
print(sub)
```

**Explanation :** 1.5 will be shown in the output. Nothing else except this, as comments have no influence on a python program.

Another example is given for you.

```
num1 = 2.5
num2 = 1.0
sub = num1 - num2
# print(sub)
```

**Explanation:** It won't print anything as we put a # before print() function. The line works as a comment and hence it won't be executed.

## 1.7 Keywords & Built-in Modules

### 1.7.1 Keywords

Python keywords, also known as “**reserved words**” are a set of built in words that upgrade the program codes. Each keyword has its own unique characteristics. For example, “in” and “not in” keywords check whether an object is present inside another set of objects or not. “if”, “else” and “elif” (else if) are used to build a conditional statement. To check the list of keywords, we can type the following lines in our code editor.

```
import keyword
```

```
keyword.kwlist
```

Procedural programming generally deals with **if, else, elif, break, pass, continue, in, not in, for** keywords. The common keywords are given below. (Collected from Google Colaboratory, Python 3.7)

and	as	assert	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	True	False	None		

Figure 1.2: Table of Python reserved keywords

Note that, as mentioned before, keywords can’t be used as variable names.

## 1.7.2 Built-in Modules

Python has a large collection of built-in modules that performs specific tasks to write a program a bit effortlessly and puts the programmers' struggle to an ease. For instance, math module, os module, random module, datetime module etc. These modules can be accessed from Python Standard Library. To use a built-in module, we need to import the module first. Write `import module_name` on the code editor to do so. Our discussion is restricted to the math module here. To access math modules, type **`import math`** in your code editor. Below some useful math modules are described.

### Math Methods

`math.acos()` : Returns the arc cosine (cosine inverse) of a number.

`math.cos()` : Returns the cosine of a number.

`math.acosh()`: Returns the inverse hyperbolic cosine of a number.

`math.asin()`: Returns the arc sine of a number.

`math.sin()`: Returns the sine of a number.

`math.asinh()`: Returns the inverse hyperbolic sine of a number.

`math.atan()`: Returns the arc tangent of a number in radian.

`math.tan()`: Returns the tangent a number.

`math.atan2()`: Returns the arc tangent of y/x in radians.

`math.atanh()`: Returns the inverse hyperbolic tangent of a number.

`math.degrees()` : Converts an angle from **radians** to **degrees**.

`math.fabs()` : Returns the absolute value of a number.

`math.floor()` : Rounds a number to its nearest integer.

`math.fmod()`: Returns the remainder of x/y.

`math.fsum()` : Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)

`math.pow()` : Returns the value of x to the power of y

`math.radians()` : Converts a degree value into radians

`math.sqrt()` : Returns the square root of a number.

`math.factorial()`: Returns the factorial of a number.

### **Math constants**

`math.e` : Returns Euler's number (2.7182...)

`math.pi` : Returns PI (3.1415...)

Find more at: [Python math Module \(w3schools.com\)](https://www.w3schools.com/python/math.asp) This is a very good website to learn basic Python. Don't forget to view the site at least for once.

**Point to be noted:** The argument to be passed inside the parameter of trigonometric functions should be in the radian form, not in degree. The output will be yielded in degree units. On the other hand, parameters of inverse trigonometric functions take arguments and return the output as degree as well. We don't need any conversion here. But it's better to convert the input in radian for phenomena like `sin()`, `cos()` etc. The following formula can be used to radian-degree and degree-radian conversion.

1 radian = 57.296 degrees

1 degree = 0.0175 radian

## **1.8 Built-in Functions**

Python's built-in functions operate some specific operations. As an example, `print()` function returns the variable, or anything valid which is inside the `()`. Things we write inside `()` are known as "arguments." An easy way to detect a built in function is, built in functions have `()` next to the function name. For instance, **`print()`**, **`type()`**, **`input()`**, **`max()`**, **`min()`**, **`round()`**,

**chr()**, **ord()** etc. In this section, we'll learn about some common functions which are used often for writing programs and solving problems.

**1. print()** : Displays something on the output console. Example:

```
>>> print("Stealth loves Roza")
Stealth loves Roza
```

**2. input()** : This function lets the user input something. Example:

```
>>> # Taking two numbers and floor dividing the numbers.
>>> user_input1 = int(input())
>>> user_input2 = int(input())
>>> floor_divide = user_input1//user_input2
>>> print(floor_divide)
```

When we run this, a prompt appears to the user/programmer which is known as an “input prompt.” We need to write and pass our input to the box. The program will then assign the input to the variable we've decided in the first place. The more `input()` function you use, the more prompt will be given to you. In this example shown below, `int()` function is used to type cast the input into an integer, in other words, only integer type data will be valid as an input. For all other data types, the program will try to cast the data into an integer. If not possible, then it will cause an error. By default, the `input()` function takes input as strings.

```
>>> user_input = input()    #Input given here is by default a string
>>> user_input2 = input()   #Input given here is by default a string
```

It's possible to add random messages to an input prompt. The method will be described later.

**3. chr():** Takes an integer number and returns a character. The returned character is the Unicode character of the given integer. Example:

```
>>> print(chr(97))
>>> #Output: 'a'
>>> print(chr(65))
>>> # Output: 'A'
```

**4. ord():** Takes an alphabet or character and returns an integer number. Being contrast to `chr()`, `ord()` returns the Unicode integer value of the given character. Example:

```
>>> print(ord('a'))
>>> # Output: 97
>>> print(ord('A'))
>>> # Output: 65
```

Keep in mind that, you can only pass only one character as the argument of the **ord()** function.

```
>>> print(ord('AB'))
TypeError: ord() expected a character, but string of length 2 found
```

**5. max():** Returns the max value from several values.

```
>>> #Example:
>>> print(max(1,4,5,8))
8
```

**6. min():** Returns the minimum value from several values.

```
>>> #Example:
>>> print(min(1,4,5,8))
1
```

**7. round():** It determines how many digits of a float number can be returned after the decimal separator. Example:

```
>>> print(round(1.2379,2))
>>> # Output: 1.23
>>> print(round(2.85789546,4))
>>> # Output: 2.8579
```

Note that, if the last digit of the resultant number is greater than 4, then while returning the number in the output console, python may add +1 to the digit. Check the second example, the

output is printed 2.8579 instead of 2.8578. But it did not make any change while returning 1.23 at the first syntax.

**8. type()** : This function tells the initials of the type/class of a specific value or object. Example:

```
>>> print(type(4))
>>> # Output : <class 'int'>
>>> print(type(4>3))
>>> # Output: <class 'bool'>
>>> print(type('Ghost'))
>>> # Output: <class 'str'>
```

Here, int, bool and str represent integer, boolean and string respectively.

**9.split()** : split() function can separate a sequence which is merged with a specific character. After separating the sequence, the separated portions are placed inside a list as items.

```
>>> fruits = 'Apple-Banana-Mango'.split("-")
>>> print(fruits)
['Apple', 'Banana', 'Mango']
```

It's not like the **split()** function only works for special characters. Any character we pass as an argument to the function, the function will work in the same way.

```
>>> fruits = 'ApplexBananaxManxgo'.split("x")
>>> print(fruits)
['Apple', 'Banana', 'Man', 'go']
```

In the argument, don't forget to put the character inside a double or single quotation mark. If the character isn't given as an argument then the whole string will be converted into a list item.

```
>>> fruits = 'Apple-Banana-Mango'.split(" ")
>>> print(fruits)
['Apple-Banana-Mango']
```



10. `strip()` : It removes the first and last characters from a string data type. We need to specify the character. If the character exists in the first and last position or either in the first or in the last position of the string, then it is removed.

#### **Example on `strip()` -01**

```
>>> var1 = ',Ocean,'
>>> var2 = var1.strip(',')
>>> print(var2)
Ocean
```

#### **Example on `strip()` -02**

```
>>> var1 = ',Ocean'
>>> print(var1.strip(','))
Ocean
```

#### **Example on `strip()` -03**

```
>>> var1 = ' Ocean '
>>> print(var1.strip())
Ocean
```

Example 03 is a miscellaneous case. To remove the spaces, it's unnecessary to specify the whitespace in the parameters of `strip()` function. For other cases, we need to pass the string argument to the parameter to make `strip()` function work.

### **1.7.3 Input prompt and `format()` function**

Check the topic of `input()` function again. When we run a code with `input()` function the program lets us input our desired data. But it doesn't show us any message. Imagine writing a program where a soldier needs to input a weapon name and manufacturer to know if the weapon is available in the armories. Now, if we develop a program that lets the user give an input but he doesn't know in which box he has to input weapon name and manufacturer as we did not show him the way. So, we must add a message to the input prompt while using the

`input()` function to make the user understand what he/she is supposed to input here.

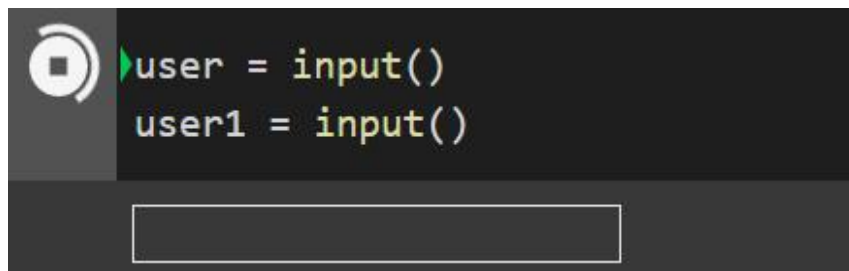


Image 1.9.a: Input prompt without any message to the user

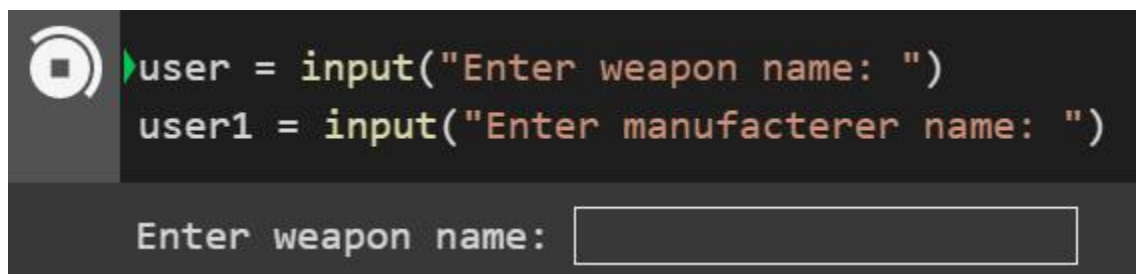


Image 1.9.b: Input prompt with a message to the user

Same formatting process can be applied for the **print()** function too. We can't always let something print without any additional words, messages or sentences. Suppose a user wants to sum two numbers and then print it.

```
>>> num1 = 6
>>> num2 = 4
>>> print(num1+num2)
```

```
>>> # After execution, 10 will be printed on the console. Now let's
edit it and make it better.
```

```
>>> num1 = 6
>>> num2 = 4
>>> print("Sum of the two numbers is",(num1+num2))
```

Now the output is: Sum of the two numbers is 10. Doesn't it sound better? Yes.

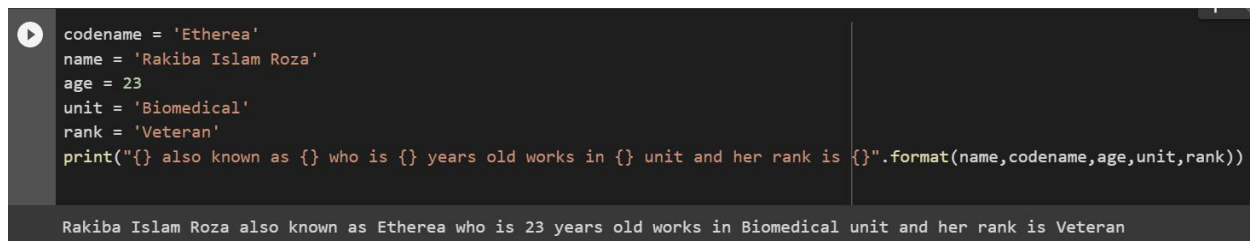
If we ever need to write a large sentence with multiple variables to be formatted then we can use **format()** function.

```
>>> codename = 'Etherea'
>>> name = 'Rakiba Islam Roza'
>>> age = 23
>>> unit = 'Biomedical'
>>> rank = 'Veteran'
>>> print("{} also known as {} who is {} years old works in {} unit
and her rank is {}".format(name,codename,age,unit,rank))
```

**Output:** Rakiba Islam Roza also known as Etherea who is 23 years old works in Biomedical unit and her rank is Veteran

The `format()` function lets us make a space (by using parentheses `{}`) for variables where the values will be placed. While writing the arguments of the **`format()`** function, we need to make sure we're writing the variable names in a proper sequence according to our will. Otherwise, the output may not seem to be desired.

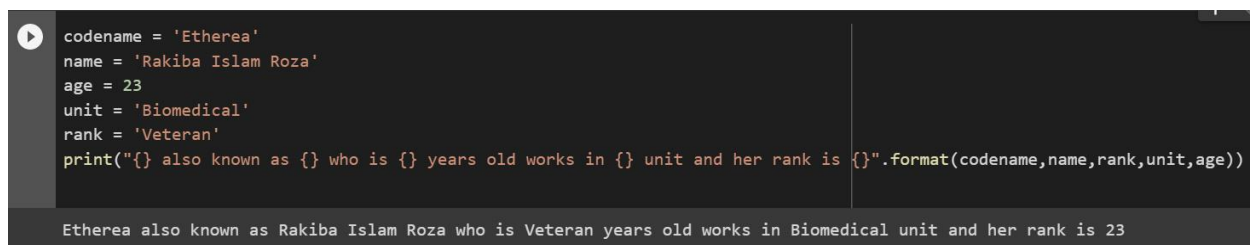
Check the previous code. What if we change the sequence of the variables?



```
codename = 'Etherea'
name = 'Rakiba Islam Roza'
age = 23
unit = 'Biomedical'
rank = 'Veteran'
print("{} also known as {} who is {} years old works in {} unit and her rank is {}".format(name,codename,age,unit,rank))
```

Rakiba Islam Roza also known as Etherea who is 23 years old works in Biomedical unit and her rank is Veteran

Image 1.9.c: Correct sequence of variables as arguments



```
codename = 'Etherea'
name = 'Rakiba Islam Roza'
age = 23
unit = 'Biomedical'
rank = 'Veteran'
print("{} also known as {} who is {} years old works in {} unit and her rank is {}".format(codename,name,rank,unit,age))
```

Etherea also known as Rakiba Islam Roza who is Veteran years old works in Biomedical unit and her rank is 23

Image 1.9.d: Wrong sequence of variables as arguments

## 1.8 Conditional Statements

Conditional statements decide whether a boolean expression is True or False and the codes under the block of statement should be executed or not. An easier way to understand conditional statements is to translate the statement into an easy to understand oral language. For example :

```
>>> var1 = int(input())
>>> var2 = int(input())
>>> if var1 > var2:
...     print("Proceed")
>>> else :
...     print("Don't proceed")
```

Let's translate from line 3. Its written if var1>var2:

so, we can say “if var1 is greater than var2 then the program should print Proceed. Otherwise, it will not print proceed (rather, it will print Don’t proceed).

The keywords **if**, **else**, **elif**(else if), **and**, **or**, **in**, **not in** etc. are used to write a conditional statement. Among them, **if,else,elif, and, or** are most used.

Example 1.1 Printing the relationships between Stealth and some random persons.

```
user_input = input("Enter a name: ")
if user_input == 'Roza' or 'Etherea' :
    print("Stealth's love")
elif user_input == 'BlueViking' or 'Prince' :
    print("Stealth's besty")
elif user_input == 'Ghost' or 'TM Shafiq' :
    print("Stealth's close friend")
else :
    print("Unknown")
```

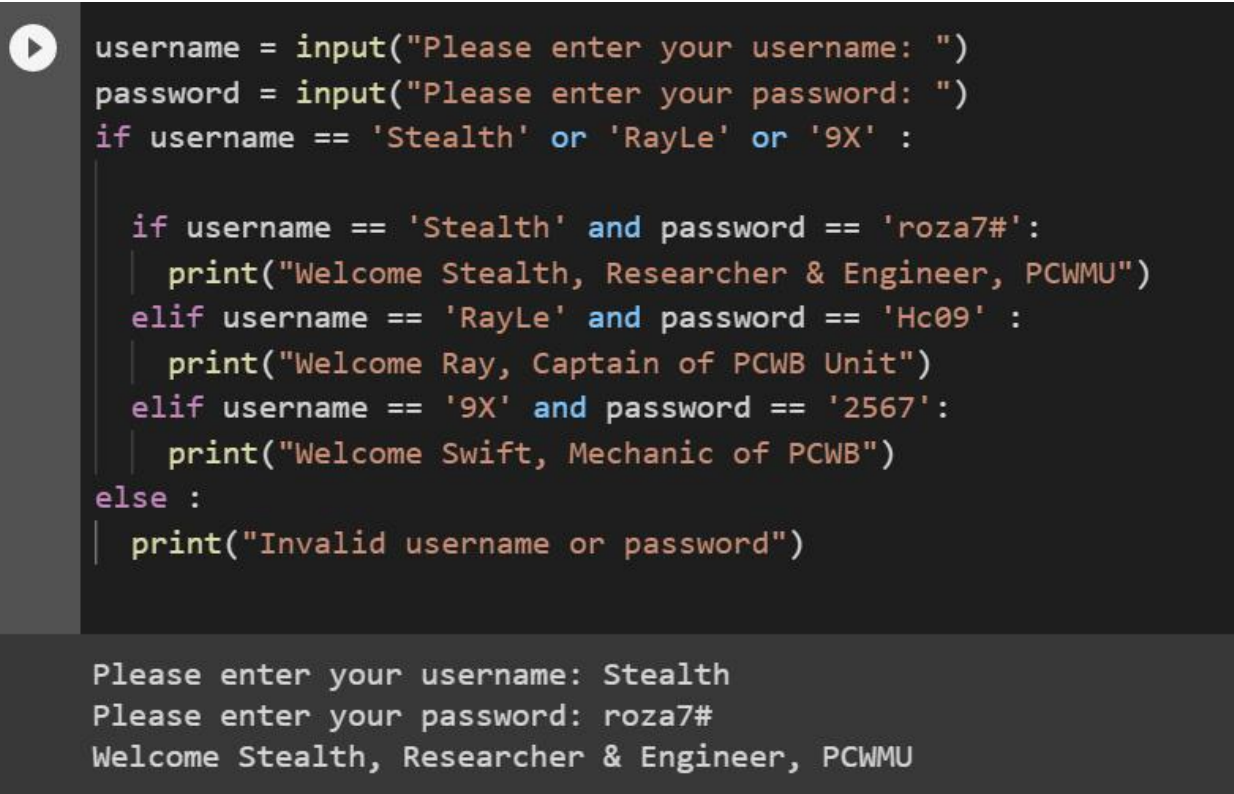
The program written above takes a person’s name and shows his/her relationship with another person named Stealth. Here, Roza and Etherea stands for the same person, the first name

denotes the nickname and the second name denotes the codename. This name pattern is followed for the other names as well. So, after an user inputs his/her name or codename the program checks whether the name is 'Roza' or 'Etherea' or something else. If the name is either 'Roza' or 'Etherea' then the program will print Roza/Etherea's relationship with Stealth. (Stealth's love). Then, next conditional statement : If the name which is given by the user isn't Roza and not even Etherea then the program will go to the next syntax and check the conditional statement of that syntax. There, it will find elif. If a conditional statement with "if" isn't True then we can use elif to command that "the program should check this elif statement and see whether the statement is true or not. If not true, then the program should skip this statement as well and move to the next one." So, if user inputs "Ghost" or "TM Shafiq" instead of "Roza" or "Etherea" then the program will print "Stealth's close friend." Just in the same manner, the program will show "Stealth's besty" as output if the user writes either 'BlueViking' or 'Prince.' If the user inputs a name/codename which is not even Etherea, BlueViking and Ghost then the program will be unable to recognize the person and mark him as an unknown person whose relationship with Stealth may be doubtful. If a True block is executed, then the other else block is excluded. This phenomenon is called unary selection. The appearance of several conditional statements one after another is known as chained/ladder conditionals.

Example 1.2 Writing a Python script of a maintenance server for Pacifia Corporation warbots that can be accessed with the correct username and password.

```
username = input("Please enter your username: ")
password = input("Please enter your password: ")

if username == 'Stealth' or 'RayLe' or '9X' :
    if username == 'Stealth' and password == 'roza7#':
        print("Welcome Stealth, Researcher & Engineer, PCWMU")
    elif username == 'RayLe' and password == 'Hc09' :
        print("Welcome Ray, Captain of PCWB Unit")
    elif username == '9X' and password == '2567':
        print("Welcome Swift, Mechanic of PCWB")
else :
    print("Invalid username or password")
```

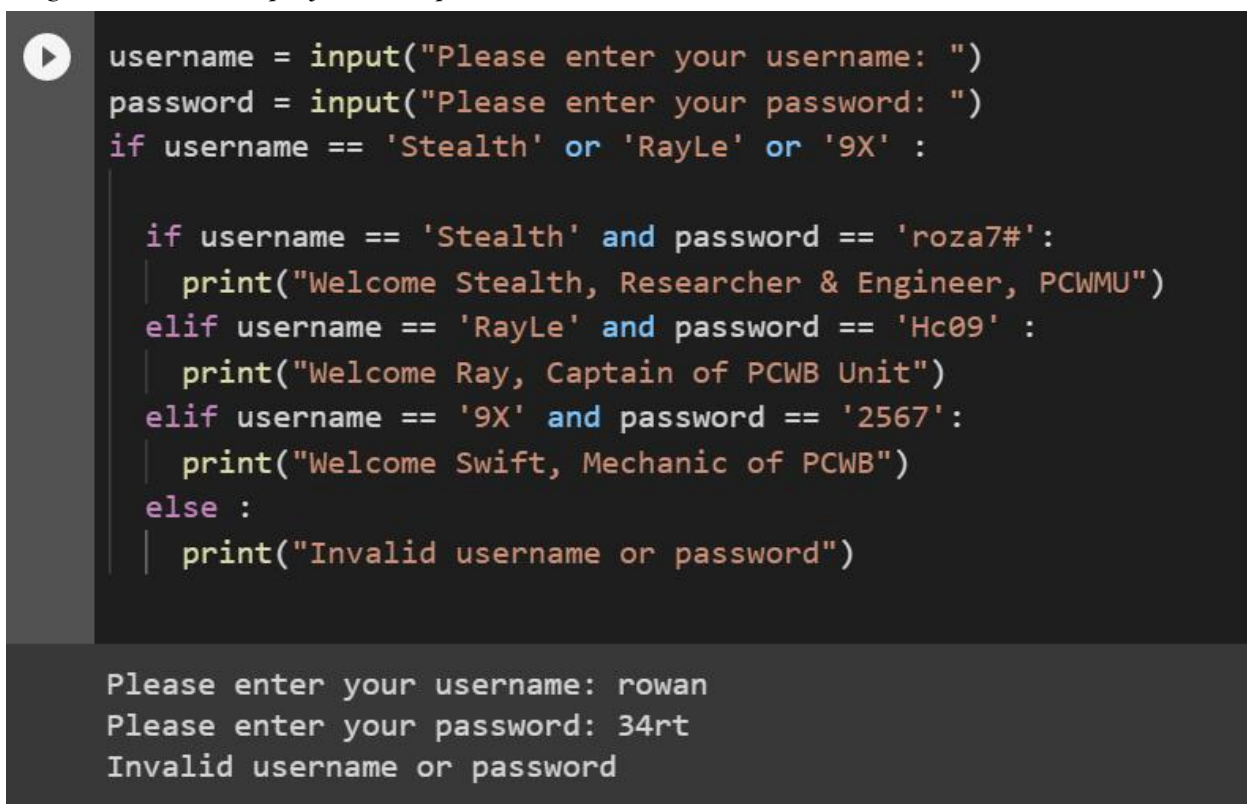
A screenshot of a Python script execution. The script prompts for a username and password. It checks if the username is 'Stealth', 'RayLe', or '9X'. If the username is 'Stealth' and the password is 'roza7#', it prints a welcome message for a researcher. If the username is 'RayLe' and the password is 'Hc09', it prints a welcome message for a captain. If the username is '9X' and the password is '2567', it prints a welcome message for a mechanic. Otherwise, it prints an invalid login message. The output shows a successful login for 'Stealth' with password 'roza7#'.

```
username = input("Please enter your username: ")
password = input("Please enter your password: ")
if username == 'Stealth' or 'RayLe' or '9X' :

    if username == 'Stealth' and password == 'roza7#':
        print("Welcome Stealth, Researcher & Engineer, PCWMU")
    elif username == 'RayLe' and password == 'Hc09' :
        print("Welcome Ray, Captain of PCWB Unit")
    elif username == '9X' and password == '2567':
        print("Welcome Swift, Mechanic of PCWB")
else :
    print("Invalid username or password")
```

Please enter your username: Stealth  
Please enter your password: roza7#  
Welcome Stealth, Researcher & Engineer, PCWMU

Image 1.10.a: Shown output for valid inputs

A screenshot of the same Python script execution as in Image 1.10.a, but with invalid input. The user enters 'rowan' as the username and '34rt' as the password. Since these do not match any of the predefined credentials, the script prints 'Invalid username or password'.

```
username = input("Please enter your username: ")
password = input("Please enter your password: ")
if username == 'Stealth' or 'RayLe' or '9X' :

    if username == 'Stealth' and password == 'roza7#':
        print("Welcome Stealth, Researcher & Engineer, PCWMU")
    elif username == 'RayLe' and password == 'Hc09' :
        print("Welcome Ray, Captain of PCWB Unit")
    elif username == '9X' and password == '2567':
        print("Welcome Swift, Mechanic of PCWB")
else :
    print("Invalid username or password")
```

Please enter your username: rowan  
Please enter your password: 34rt  
Invalid username or password

Image 1.10.b: Shown output for invalid inputs

## Algorithm Visualization:

First of all, the program will ask the user to input his username and password. If the user is someone who has the access to log in to the Pacifica Corporation Warbots maintenance server then the program would let him in.

There are three people who have the access to log in to the server. Their codenames are Stealth, RayLe and 9X.

**Line 01 and Line 02 :** The program takes input from users in these lines.

**Line 03 :** The program checks if the user who is trying to log in is someone from Stealth, RayLe or 9X.

**Line 04:** Intentionally left blank. This line won't be executed.

**Line 05:** If the user is "Stealth" and he gives the correct password (roza7#) then the server will welcome him (See line 06.)

**Line 07:** If the user isn't "Stealth" then it must be someone else from the trio. So the program will move to line 07's elif statement. Under the statement the program would verify the user like it did at Line 05. The whole program till line 10 will check and verify the users in this way.

**Line 11 :** There we can find a conditional statement with else condition. According to the statement, if the user isn't someone from Stealth, RayLe and 9X or the user types his password incorrectly then the codes under the outer if block won't be executed, so do the inner if blocks. As a result, the server won't let him login and send a message - "Invalid username or password." This phenomenon of **if** inside another **if** is known as a nested condition. Nested conditions can be created for else and elif conditions as well. If the first condition (boolean expression of outer block) is True, only then the interpreter will go to the code blocks under the second if block and check whether it's True or False.

**Structure of a conditional statement:** keyword + "boolean expression" + ":" +  
"indentation in the next line"

Example

```
>>> if 3<4:
...     print("True")
```

Here, ‘if’ is the keyword, ‘3<4’ is the boolean statement and “:” denotes that the conditional statement is done written, four spaces before beginning of second lines are the indentations. By using proper indentation, we make the interpreter understand that the code blocks are under the conditional statement. Indentation refers to the *scope* of a conditional statement.

**Note 1:** We don’t need to write a boolean expression beside an **else** statement. We need to simply write else: and then script the codes under the else block. If the codes inside the “if” block aren’t executed then it will be executed inside the else block.

### Example

```
>>> user1 = int(input("Enter a number: "))
>>> user2 = int(input("Enter a number: "))
>>> if user1 > user2:
...     print("True")
>>> else :
...     print("False")
```

If user1 is less than use2 then **False** will be printed on the console.  
On the other hand, if user1 is greater than user2 then **True** will be printed on the console.

**Note 2:** “if” and “else” work in pairs. An “else” statement can’t exist without “if” but conditional statements with “if” can exist without “else.”

### Example

```
>>> number = 70

>>> if number > 90:
...     print("True1")
>>> if number > 80:
...     print("True2")
```



```
>>> if number>60:
...     print("True3")
>>> else:
...     print("False")
```

The last conditional statement with the **else** keyword is a pair of its previous “if” statement, so codes inside the “else” block won’t be executed even if the first two “if” conditional statements of this program are false(wrong). If we add another statement after the first conditional statement using else and give a message to it then it will be printed.

```
>>> number = 70
>>> if number > 90:
...     print("True1")
>>> else :
...     print("False1")
>>> if number > 80:
...     print("True2")
>>> if number>60:
...     print("True3")
>>> else:
...     print("False2")
False1
True2
```

***Nested conditional statement :*** The example 2 shown above has a unique attribute. It began with an “if statement” and also there are a bunch of more “if statements” inside the first if statement. This is called nested if. Similar to this, sometimes we might feel the necessity of using nested else.

Another example of nested if and nested else:

```
>>> user1 = int(input("Enter a number: "))
>>> user2 = int(input("Enter another number: "))
>>> if user1 and user2 > 0:
...     print(user1+user)
>>>     if user1 > user2 :
...         print(user1,"is greater than", user2)
```

```

>>> if user1 < user2 :
...     print(user1,"is less than", user2)
>>> else :
...     print("Cannot input 0")
>>> if user1 == 0:
...     print("user1 = 0")
>>> if user2 == 0:
...     print("user2 = 0")

```

**Data types:** Notice carefully, the data types we had to use in these codes are “string.” Because username and passwords are usually combinations of alphabets, numbers and special characters. So while we were assigning data into username and password variables inside conditional statements, every name was put inside quotation marks. Another information, Python’s **input()** function **converts** every given input into **string by default**. We do not need to write `str(input())` here. But if anyone wishes to, then it’s fine. It won’t hamper the code from execution. That’s up to the user’s choice.

**Indentation rules:** We can’t begin writing our codes without any space gap from the next line after the conditional statement. An indentation block is required. The space gap/indentation refers to the scope of a conditional statement, a range that determines certain scripts belong to the “if” block and these script codes will be executed only and only if the boolean expression of “if” block is True.

```

>>> if 3<4:
...     print("True")      #Correct
>>> if 3<4:
    print("True")          #Incorrect

```

We must focus under which conditional statement we’re editing codes. Indentation controls everything regarding this issue. But you do not need to worry much about it. A good thing is, most IDEs and code editors automatically create an indentation when it's needed.

***Difference between “or” and “and” keyword: When to use or? When to use and?***

**Use of “or” keyword :** Check line 03. When the program needed to check who the user is then we used or. “or” keyword makes the conditional statement in a way that if any of the boolean expressions is True then the code will be executed/go to the lines under the conditional statement. Translate line 03 into English. It will be like : “If the user is Stealth or RayLe or 9X then the program will check the code blocks under the statement.” This means, we just need to satisfy one criteria to execute the codes under the conditional statement.

**Use of “and” keyword :** Check line 05, 07 and 09. When the program needed to verify username and password both, not the only one thing, there we used “and” keyword. Means, we need to satisfy both conditions to move to the codes under the conditional statement. An user must type his username AND password correctly to log in to a server, right? Should the server let the user enter the server if he/she only types the username or the password correctly? Obviously it shouldn't.

## 1.9 Operator Precedence

Order of operations (also known as **operator precedence**) refers to a set of rules to determine how a mathematical expression should be evaluated. Operator precedence is implemented when we need to work with compound expressions; expression that contains multiple operators and mathematical operations is known as a compound expression. For example,  $2+3*24/6-2+3$  is a compound expression. Python follows the same precedence rule like the other sections of mathematics. The order of operations in mathematics are given below:

Parentheses > Exponentiation > Multiplication, division > Modulus, floor division > Addition, subtraction

Comma separated terms have the same precedence.

1. Exponentiation: The precedence of exponentiation is higher than others. So calculate the numbers operated with the powers first.
2. Multiplication and division: If you find multiplication and division both in an expression, operate the division first. Then proceed for multiplication.

3. Addition and subtraction: The precedence for addition and subtraction is the same. You can do addition first if you want, or you can do subtraction as well.
4. Modulus: Precedence of modulus comes after division and multiplication.

The relation between the operands starts from left to right. Separate the expression into smaller parts if the expression is too big. We can separate the expression at the point where there is either + or - since these have the lowest precedence and we do these processes in the end. Lets understand by trying to determine the resultant of:  $10*4\%3+5-2*4**2$

Step 1:  $10*4 = 40$

Step 2:  $40\%3 = 1$  (store)

Step 3:  $4**2 = 16$  (store1)

Step 4:  $2*16$  (store1) = 32

Step 5:  $1(\text{store})+5-32 = -26$  (Answer)

We can write the same expression in this way:  $(10*4)\%3+5-(2*(4**2))$ . The parentheses denote the precedence more clearly and the expression is now easier to evaluate. If we evaluate the expressions inside the parentheses, gradually we'll get the final result.

More examples are shown below.

Example a:  $2*3 + 4/5 -1 = 5.8$

Example b:  $((2*3)+(4/5)-1) = 5.8$

Example c:  $(2*(3+4))/(5-1) = 3.5$

Example d:  $3+4-(2+7/3)/3.0\%2 = 5.55$

## Exercises

**Objective 01.** Write a Python script to make a simple calculator that can operate the calculation of two numbers only. The numbers can be either integers or floats. You need to take three inputs. One is for the operator (+, -, \*, /) and the other two inputs are for the numbers.

### Sample Input & Output 1

Input	Output
+ 10 20	30
- 10.5 5	5.5
* 2 4	8

**Objective 02.** Write a Python program which will take an input from the user and display whether the number is an even number, odd number or a float number on the console.

### Sample Input and Output

Input	Output
-------	--------

2	Even
3	Odd
2.0	Float

**Objective 03.** Write the Python code of a program that reads an integer, and prints the integer if it is a multiple of **either 2 or 5 but not both**. If the number is a multiple of 2 and 5 both then print “Not a multiple we want”.

Sample Input & Output

Input	Output
15	15
10	Not a multiple we want

**Objective 04.** Write a Python program that takes three inputs from the user which are the length of the three legs of a triangle (input the largest number first) and prints whether the numbers are Pythagorean triplet or not.

Sample Input & Output

Input	Output
25 7 24	They are Pythagorean Triplet
5 5 5	They aren't Pythagorean Triplet

**Objective 05.** Given numbers: 56, 45, 67, 80

Write a Python code to:

a) Print the sum of the numbers

b) Print the average of the numbers

Format your output like: “Sum of the numbers are \_ and the average of the numbers is \_”

Output
Sum of the numbers is 248 and the average of the numbers is 62

**Objective 06.** Write a Python script that will take two inputs from the user. If the second input exists inside the first string input then the program will print “True”. Otherwise, it will print “False.”

Sample Input & Output

Input	Output
abcdef b	True
qwerty ert	True
yui yi	False

**Objective 07.** Write the Python code of a program that takes the values of a, b, c respectively from a quadratic equation and tells the value of the determinant D and the nature of the equation’s root. [Standard quadratic equation =  $ax^2 + by + c$  and  $D = b^2 - 4ac$ ]

If  $D = 0$  then print “Real and equal”

If  $D < 0$  then print “Imaginary”

If  $D > 0$  then print “Real and unequal”

Sample Input & Output

Input	Output
3 -10 3	Real and unequal
3 4 6	Imaginary

**Objective 08.** Write a Python program that takes 6 inputs (components of a vector in unit vector notation on a three dimensional coordinate plane), creates two vector quantities from the inputs and prints the angle between the vectors. *Special case:* If the angle is 0 degree, then print “The vectors are parallel to each other” and if the angle is 90 degrees, then the program should print “The vectors are perpendicular to each other.”

Formula:  $A \cdot B = AB \cos \theta$

Where the bold **A** and **B** are the vectors, unbold A and B are their magnitudes and  $\theta$  is the angle between them. In a unit vector notation,

$$\mathbf{A} = A_x \mathbf{i} + A_y \mathbf{j} + A_z \mathbf{k}$$

$$\mathbf{B} = B_x \mathbf{i} + B_y \mathbf{j} + B_z \mathbf{k}$$

$$A \cdot B = \sqrt{A_x^2 + A_y^2 + A_z^2} \cdot \sqrt{B_x^2 + B_y^2 + B_z^2}$$

Here  $A_x, A_y, A_z, B_x, B_y, B_z$  are the vector components.

Sample Input & Output

Input	Output
$A_x$ : 3 $A_y$ : -1	



$A_z: -2$ $B_x: -1$ $B_y: -3$ $B_z: 4$	The angle between the vectors is 114.792 degrees.
---	---

Test your program on the following vectors too.

(a)  $\mathbf{A} = 6\mathbf{i} - 3\mathbf{j} + 2\mathbf{k}$  and  $\mathbf{B} = 2\mathbf{i} + 2\mathbf{j} + \mathbf{k}$

(b)  $\mathbf{A} = 2\mathbf{i} + \mathbf{j} - 2\mathbf{k}$  and  $\mathbf{B} = 5\mathbf{i} - 4\mathbf{j} + \mathbf{k}$

(c)  $\mathbf{A} = \mathbf{i} + \mathbf{j} + \mathbf{k}$  and  $\mathbf{B} = 5\mathbf{i} + 5\mathbf{j} + 5\mathbf{k}$

**Objective 09.** Imagine you work in a multinational militia company named Pacifica Corporation as a weapon maintenance engineer in the Weapons and Munitions Unit. The corporation soldiers are in need of several machine guns for close range battles which recoil is not more than 2 m/s. Write a Python program, which will take the mass of the gun ( $M$ ) and its bullet ( $m$ ), speed of the gun's bullet ( $v$ ) as inputs and display whether the weapon is usable in warfield or not. If the recoil velocity exceeds 4 m/s then put a warning message suggesting not to use that gun.

Formula: Recoil velocity  $V = (m/M)v$

Sample Input & Output

Input	Output
$M = 10 \text{ kg}$ $m = 0.04 \text{ kg}$ $v = 962 \text{ m/s}$	The recoil velocity is 3.84 m/s. This weapon is unusable.
$M = 7 \text{ kg}$ $m = 0.02$ $v = 700 \text{ m/s}$	The recoil velocity is 2 m/s. This weapon is usable.
$M = 8 \text{ kg}$	6 m/s+ recoil. Excessively high recoil

$m = 0.05 \text{ kg}$ $v = 1020 \text{ m/s}$	warning! This is a hybrid weapon and suggested not to use in close range battles.
---	---

**Objective 10.** Here's something very annoying. Your friend is preparing the lab report for PHY111 and she finds out the following equation.

Moment of inertia of the flywheel about its axis of rotation,

$$I = \frac{(ght^2 - 8\pi^2 n_2 r^2)M}{8\pi^2 (n_2^2 + n_1 n_2)}$$

You reminded how struggling the whole thing was, doing the same calculation over and over again was insane. It took a lot of time and several mistakes occurred. So you hit upon a plan to help her. You're going to write a Python program that will take the necessary variables and constants of the equation from her and then display the moment of inertia on the console.  
[Value of h is in cm.  $g = 981 \text{ cm/s}^2$  ]

## Chapter 02: Loops

Python loops are used to iterate objects in a sequential manner and it can be used to increment or decrement numbers or objects. An easy synonym of iterate, increment and decrement is “repeat”, “increase” and “decrease” respectively. Sometimes we may need to iterate numbers or items to get our desired output. Suppose, we have to sum up the first ten numbers (1 to 10). It would be arduous to write the same codes again and again to get the result. This is where the loops can help us.

Python has two types of loops. They are known as **for** and **while** loop. We use the keywords “for” and “while” to create a loop and can process iteration via the loop.

### 2.1 For loop

#### **Attributes:**

- a) A for loop is created by using the “**for**” keyword.
- b) Usually **range()** function and keyword ‘**in**’ is used while creating a for loop.
- c) For loop can be applied on numbers, string, list and other iterables.

#### **The range() function :**

The function range() is used to generate numbers that follow the pattern which is created by the user. Range function has three parameters that create the pattern. Each parameter takes only one argument which must be in integer data type.

Syntax: range(start,end,step).

**Start :** The number from where the counting begins.

**End:** The number before what the counting ends. It depends on the step argument. If we input 10 as the argument of end parameter and 1 as step parameter then the

increment will be stopped at 9. For the same case if we input 2 as step parameter then the increment will be stopped at 8 and the process goes like this. Check the examples for clarification. (See examples from example 2.1.1)

**Step:** The step that denotes the increment or decrement step.

By default, the starting argument is 0 and step is +1. End argument must be given by the user.

### Usage of in keyword in for loops

We use in keyword to mention where the for loop should run.

```
for iter in range(1,9):  
    print(iter)
```

This for loop generates numbers that are in range between 1 to 9. Here, iter is the iteration variable which consequently stores the generated number per iteration. Now, look at this for loop shown below:

```
user = 'Soap'  
  
for iter in user:  
    print(iter)
```

This time, the for loop doesn't generate numbers as the loop is running over the user variable, not in the range() function. The user variable contains some alphabets. Hence, iter will store the alphabets (S,o,a,p) per iteration.

### Structure of a for loop

A for loop has an 'iteration variable' that keeps changing its value based on a sequence given by the user. The sequence can be made by using a string, list, tuples or numbers. While dealing with numbers, we use the range() function. In other cases like sequence data types, we use "in

sequence\_data\_types” formation or evaluate their length to use it as the range() function’s argument.

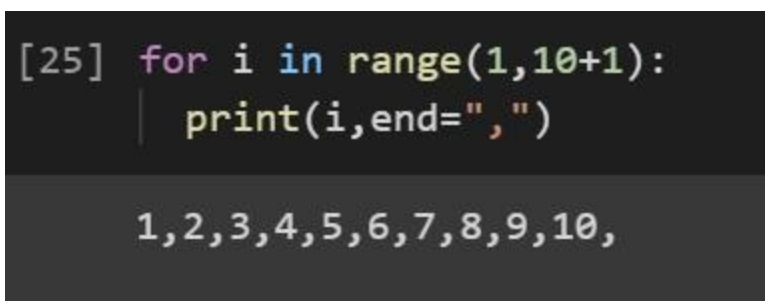
```
for iteration_variable in
"sequence/collections/number_range" :

    *repetitive work/block of statements*
*codes outside the for loop (optional)*
```

Figure 2.1: Structure of a for loop

Some examples are given below to clarify the concept. We'll also see the examples of the usage of for loop.

*Example 2.1:* Print 1 to 10 in the console.



```
[25] for i in range(1,10+1):
      print(i,end=",")

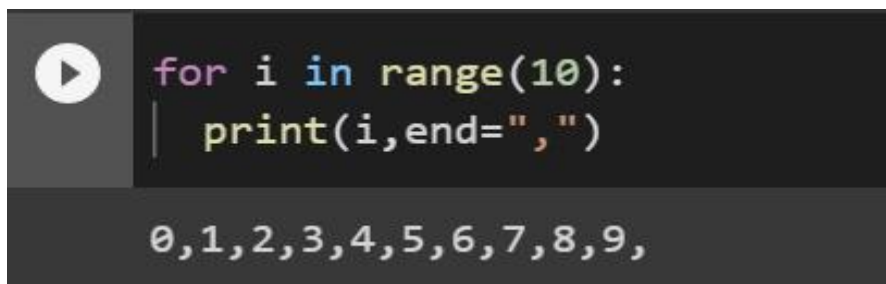
1,2,3,4,5,6,7,8,9,10,
```

Image 2.1.a: Printing the numbers 1 to 10 using a for loop.

Here *i* is the iteration variable, generally means iteration, is something that helps us to iterate values. The meaning of the syntax is like: “The user given range is (1,10+1) and *i* will be iterated within this range and its value will be changed per iteration. After the iteration is completed, the console will show every value of *i*.” Note that, we wrote ‘10+1’ in end

parameter because if we've written 10 then the iteration would have been stopped at 9 and the printed numbers in the console would have been 1 to 9.

*Example 2.2*: Print 0 to 9 in output console.

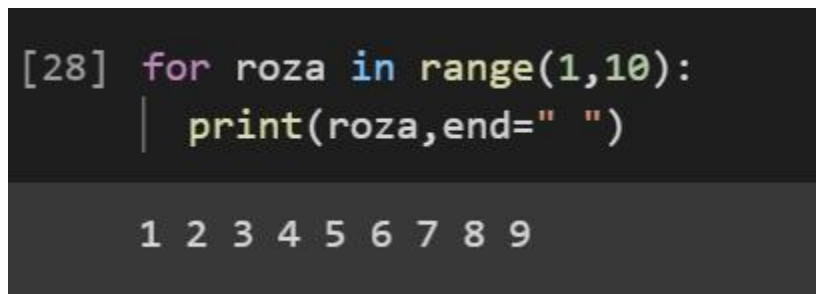
A screenshot of a code editor with a dark background. On the left, there is a play button icon. The code is written in a syntax-highlighted font: `for i in range(10):` on the first line and `print(i,end=",")` on the second line. Below the code, the output is displayed in a lighter font: `0,1,2,3,4,5,6,7,8,9,`.

```
for i in range(10):  
    print(i,end=",")  
  
0,1,2,3,4,5,6,7,8,9,
```

Image 2.1.b: Printing 0 to 9 using a for loop

If we put only an argument then by default our input becomes the end parameter. Start and step parameters become 0 and 1 respectively. As the end parameter is 10 and step is 1 so the console printed 0 to 9.

*Example 2.3*: Print 1 to 9 in output console.

A screenshot of a code editor with a dark background. On the left, there is a prompt `[28]`. The code is written in a syntax-highlighted font: `for roza in range(1,10):` on the first line and `print(roza,end=" ")` on the second line. Below the code, the output is displayed in a lighter font: `1 2 3 4 5 6 7 8 9`.

```
[28] for roza in range(1,10):  
      print(roza,end=" ")  
  
1 2 3 4 5 6 7 8 9
```

Image 2.1.c: Printing 1 to 9 using a for loop

It's not like we can only use `i` to iterate something. We can use any other valid names following variable naming convention and rules instead of writing `i`.

*Example 2.4:* Print all the odd numbers from 1 to 20.

```
[29] for stealth in range(1,20,2):  
    | print(stealth,end=" ")  
  
1 3 5 7 9 11 13 15 17 19
```

Image 2.1.d: odd numbers between 1 to 20

Here we have given 2 in the step parameter's argument. So while incrementing, the number was being added with 2 every time. Hence the console shows every odd number from 1 to 19.

*Example 2.5:* Print the sequence: 20 18 16 14 12 10 8 6 4

```
[31] for ghost in range(20,2,-2):  
    | print(ghost,end=" ")  
  
20 18 16 14 12 10 8 6 4
```

Image 2.1.e: Sequence 20 to 4 with a gap of -2

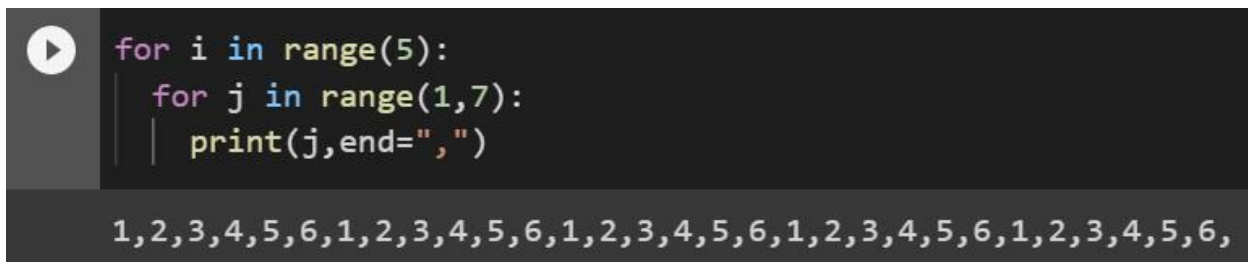
We can step backwards as well. To do so, negative integers should be passed to the step parameter..

Note: It's a bad habit to use bizarre names as variables. I used the names to show you examples only. While writing programs, avoid using irrelevant names and follow the variable naming conventions. Select a meaningful name for your variable so that it can represent the data stored inside the variable.

**About the end parameter in the print function :** The end parameter in the print function is used to add any string. By default, the print function ends with a newline. Passing the whitespace to the end parameter (end=' ') indicates that the end character has to be identified by whitespace and not a newline. We can add anything instead of whitespace as well. For that, we need to give the string inside the quotation mark.

```
>>> print("A",end="hehe")
>>> print("B")
AheheB
```

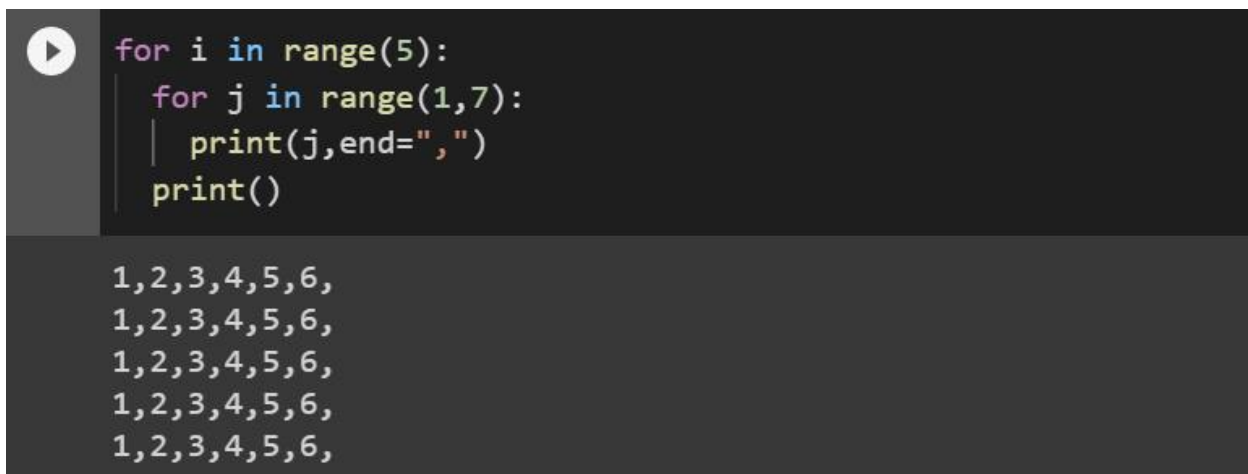
**Nested loop:** Sometimes it's needed to use a loop inside another loop. If a loop exists inside another loop then we call it a nested loop. The first loop is called outer loop and the second is called inner loop. Example:



```
for i in range(5):
    for j in range(1,7):
        print(j,end=",")
```

1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6,

Image 2.1.f: The inner loop should print the numbers 1 to 7(values of j). But since it's inside another loop (outer loop) that iterates 5 times, the inner loop prints the numbers 1 to 7(values of j) 5 times.



```
for i in range(5):
    for j in range(1,7):
        print(j,end=",")
    print()
```

1,2,3,4,5,6,  
1,2,3,4,5,6,  
1,2,3,4,5,6,  
1,2,3,4,5,6,  
1,2,3,4,5,6,

Image 2.1.g: If we want to print the values of j (numbers that are generated by the inner loop)



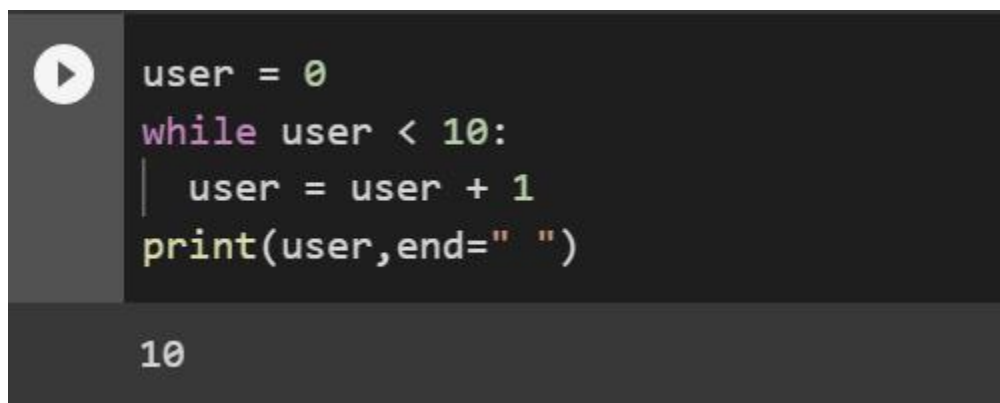
line by line, not in linear sequence, then we can simply write “print()” outside the inner for loop. This will let our job be done by creating a new line.

## 2.2 While loop

While loop usually exactly does what the for loop does, both loop iterates iterables but they both have limitations as well. Anyway, based on some given statements, a while loop iterates objects. The difference between a for and while loop is on their structure. Except this, both loops can be used to modify values. It's up to the user which loop should be used to complete a specific task. In some cases, a while loop may solve a problem easily and in some other cases, for loop can be used. But usually, if we master only one type of loop it would create no problem for us. Both kinds of loops are good to go. In this book, for loop is used to solve most of the problems as while loop may seem more difficult and bigger syntax sized than a for loop. The algorithm for while loop needs to be set manually without using range() function while we're dealing with numbers.

### Structures of a while loop

A while loop consists of five elements. A keyword (**while**), variable, boolean expression, colon and indentation. On the other hand, a for loop is made of two keywords(**for, in**) colon and indentation. It doesn't matter which loop we're using, we must maintain indentations. Anything we write outside the loop (without maintaining indentations) will not be executed. After the iteration, the last generated value will be assigned only.



```
user = 0
while user < 10:
    user = user + 1
print(user, end=" ")

10
```

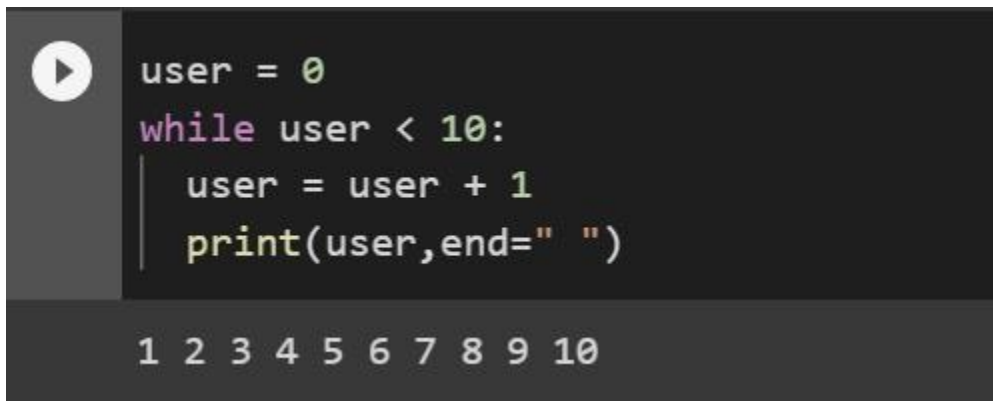
Image 2.a : Since we wrote `print()` function outside the loop (without any indentation subject to the while loop's creation line) so the console printed the last generated value by the iteration process which is 10 instead of printing 1 to 10.

The structure of a while loop is depicted below inside a code cell. See the examples for clarification.

```
#a variable to store an initial value (this value changes per
iteration)
#while condition (loop controller and termination statement)
    #codes inside the loop
    #codes inside the loop
#codes outside the loop (optional)
```

Figure 2.2: Structure of a while loop

*Example 2.6:* The following program prints the numbers 1 to 10.

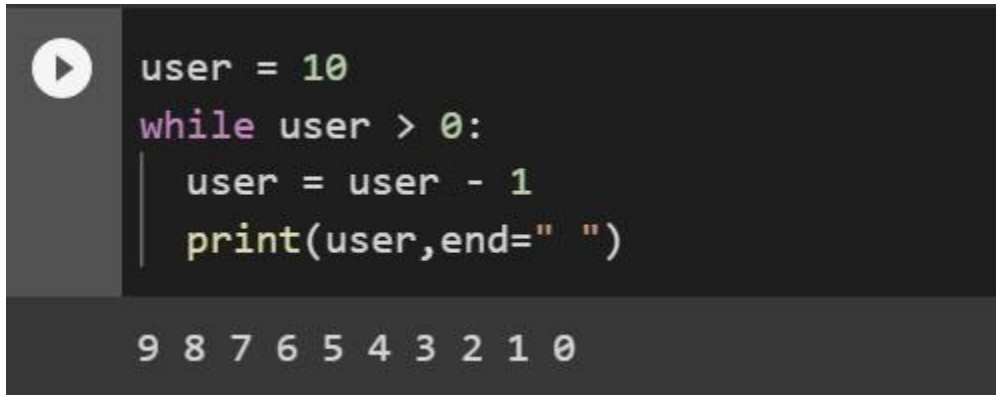


```
user = 0
while user < 10:
    user = user + 1
    print(user, end=" ")

1 2 3 4 5 6 7 8 9 10
```

Image 2.2.a: First of all, we took a variable “user” where we stored our initial number (0) from which the iteration will begin and increment the variable’s value. The second and third line together (a while loop is created) stands for: “While the user variable’s value is less than 10, the loop will keep iterating and each time the loop iterates, the value of the variable (user) will be incremented +1 (user=user+1 commands so).” So, while on its first iteration, user = 0+1 = 1; second iteration, user = 1+1 = 2; third iteration, user = 2+1 = 3 and on its tenth iteration the value becomes 10. Then if we use `print()` function **inside** the while loop (by maintaining proper **indentation**), it will print every number that was generated during the iteration process (in this example, 1 to 10).

*Example 2.7:* Print 0 to 10 numbers in reversed order.



```
user = 10
while user > 0:
    user = user - 1
    print(user, end=" ")

9 8 7 6 5 4 3 2 1 0
```

Image 2.2.b: Here our initial value is 10 which is stored in a variable called 'user'. The while loop will continue its iteration process and keep decrementing the user's value -1 while the value is greater than 0. When the value becomes 0, the loop will be stopped and generated numbers will be printed.

We need to be careful when writing the increment syntax under a while loop. It can either increment the loop controller before printing it or increment the loop controller after printing it. Examples are shown below.

```
>>> counter = 1 #counter variable is the loop controller here
>>> while counter < 10:
... counter = counter + 1
... print(counter)
2
3
4
5
6
7
8
9
10
```

The script written above will make a program that prints the numbers 2 to 10 in a row. Because the initial value of the counter is 1. After the interpreter reaches line 3, the line under the while loop, where it's told to increment the counter's value by 1. So, the counter's value will become 2 before the interpreter reaches the last line where the programmer is commanded to print the newest value of the counter. The iteration will go on like this and when the value of counter reaches 9 by incrementing, just like the previous way, counter's value becomes 10 and the interpreter prints that as well. Now, look at the following program:

```
>>> counter = 1
>>> while counter < 10:
...     print(counter)
...     counter = counter + 1
```

Unlike the previous script, this will print the numbers 1 to 9 on the console. Because we used the print() function one line before writing the increment step. The present value of counter will be printed first, then at line 4, the value of counter will be incremented and replace the old value with the newer value. This is how the value will be updated and printed.

## 2.3 Infinite Loop

While loops can create a nonstop repeat of an output. The program will never terminate and will keep repeating the same output again and again until the memory runs out. This is called *infinite loops*.

```
while True:
    print("Stealth")
```

This program will keep posting “Stealth” without any termination. The statement means, while the boolean expression is True, keep printing Stealth. Since there is no condition to terminate the loop and the condition is True forever, so the loop will never last. It will keep executing and printing the valid output which is scripted by a programmer until the memory runs out. Infinity loop has an advantage. It lets a user input as many values he wants to or to print something over and over again. As a result, if we ever need to create a program where the

user needs to input uncountable values or objects, an infinite loop can be used. Suppose, you're asked to write a Python program which will read integers from the user and when the user is done with giving the inputs, he will simply press Enter and the program will print the sum of every integer that he inputted. The solution of this problem is solved below:

```
sum = 0
while True:
    user = input("Enter number (blank to stop): ")
    if user == "":
        break
    else :
        sum = sum + int(user)
print("The sum of the numbers is",sum)
```

Another approach to solve this problem can be like this:

```
sum = 0
count = 1
while count > 0:
    user = input("Enter number: ")
    if user == "":
        break
    else :
        sum = sum + int(user)
print("The sum of the numbers is",sum)
```

A variable is assigned which value is more than 0. Then we set the boolean condition at the while loop that denotes, while 1 is greater than 0 keep continuing the loop. In the loop, we did not decrement the variable's value. So for every iteration, the value of 'count' is 1, which is greater than 0. Therefore, the loop will keep iterating and this will let the user input numbers incessantly. The loop will never stop until we set another condition to break the loop. That's what we did in line 5, when the user will press Enter (blank input) the loop will be stopped. While the user were inputting numbers (not string), the numbers will be executed under the else block and the sum of the numbers will be stored.

```
count = 1
```

```
while count > 0:
    print(count)
    count = count + 1
```

This small but ferocious program will keep generating 1 to infinite numbers within an eyewink. Because the initial value of count is already greater than 0 and still the programmer commanded to increment it by 1.

## 2.4 Early Exit

We might face several situations when breaking a loop would be necessary. Suppose, we have to print all the numbers starting from 1 to 100 but we cannot include 50 to 60 among these numbers. When the iteration will start generating 50, the loop should ignore that generated numbers to 60 and start printing from 61. Also think about some other examples, a list of male and female names are given where the female names have 'a' at the end of their name. If we want to get the male names only then we must ignore the names that end with 'a.' In this case, we will need to keep passing the loop for every name that has 'a' in the ending of a name and thus only male names will be remaining. Lets get introduced with three keywords that terminate a loop, skip a loop to next one and the other one let the loop continue on its consistent flow ignoring a step. The keywords are break, pass and continue respectively. Since these keywords have a direct impact on a loop and can terminate the loop all of a sudden, this phenomenon is known as **early exit/ early leave** from a loop.

### 2.4.1 Implementation of loop terminators in while loop

By using **break** we can stop a loop ignoring the loop terminating conditions which is written as a boolean expression beside the while keyword.

```
count = 1
while count < 11:
    count = count + 1
    if count == 5:
```

```

        break
    print(count,end=",")
2,3,4,

```

The loop is stopped at count's value 5 without reaching 10.

By using **continue** we can deny current iterations and move to the next iterations.

```

count = 1
while count < 11:
    count = count + 1
    if count == 5:
        continue
    print(count,end=",")
2,3,4,6,7,8,9,10,11,

```

The program won't print 5. But will **continue** printing numbers after 5.

Unlike the above shown loop terminators, **pass** directly skips the loops and moves to the next statements. When possible statements aren't found, pass let the loop iterate on its current flow.

```

count = 1
while count < 11:
    count = count + 1
    if count == 5:
        pass
    print(count,end=",")
2,3,4,5,6,7,8,9,10,11,

```

## 2.4.2 Implementation of loop terminators in for loop

Implementation of the **break** statement:

```

for i in range(1,11):
    print(i,end=",")
    if i == 5:
        break
1,2,3,4,5,

```

Implementation of the **continue** statement:

```
for i in range(1,11):  
    if i == 5:  
        continue  
    print(i,end=",")  
1,2,3,4,6,7,8,9,10,
```

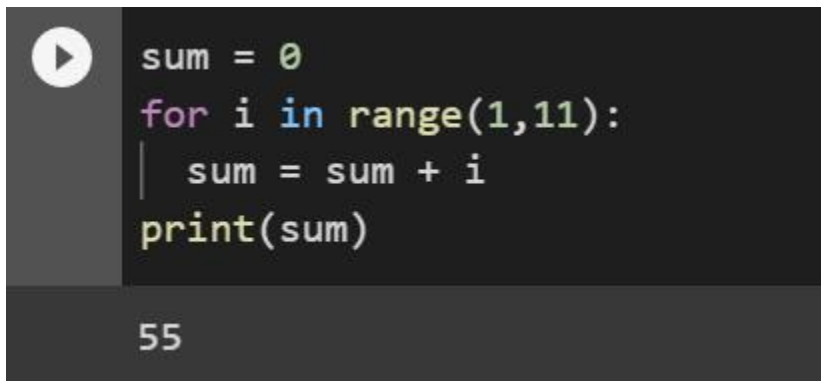
Implementation of the **pass** statement:

```
for i in range(1,11):  
    if i == 5:  
        pass  
    print(i,end=",")  
1,2,3,4,5,6,7,8,9,10,
```

## 2.5 Program Analysis

Some sample programs are given below. These programs are broken into its core parts in the algorithm visualizations to show how these programs actually work. This will clear this chapter's concept better.

a) Write a python program that can sum up to the first ten integers.

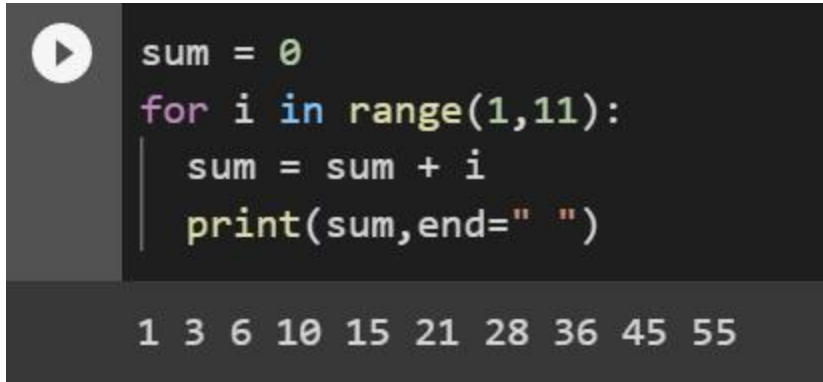


```
sum = 0  
for i in range(1,11):  
    sum = sum + i  
print(sum)
```

55

Image 2.4.a: Solution 1 for the given problem





```

sum = 0
for i in range(1,11):
    sum = sum + i
    print(sum,end=" ")

```

1 3 6 10 15 21 28 36 45 55

Image 2.4.b: Alternative solution for the given problem that visualizes step by step summation.

### Algorithm visualization:

A line by line explanation is given below to understand the problem.

**Line 1:** A variable (sum) is created and a value (0) is assigned to it and stored here. This is an initial value, it will be changed per iteration.

**Line 2:** A for loop is created with proper arguments following the given question inside range() function.

**Line 3:** Here “sum=sum+i” is written. In this line, Right Hand Side (sum) represents the variable while Left Hand Side (sum + i) represents the values. That means, “sum+i” is assigned to “sum.” It’s possible to assign changed values to a variable. By doing so, the variable’s stored value in the first place will be replaced by the last one.

For example:

```

>>> var1 = 2
>>> var1 = 3
>>> print(var1)
>>> # Output: 3

```

### (Inside The Loop)

**Line 3 to line 4:** On the loop's first iteration, based on the given arguments inside range function, the new values for sum and i will be generated.

Iteration 1:  $\text{sum} = \text{sum} + i$

means,  $\text{sum} = 0 + 1 = 1$

Iteration 2:  $\text{sum} = \text{sum} + i$

means,  $\text{sum} = 1 + 2 = 3$

Iteration 3:  $\text{sum} = \text{sum} + i$

means,  $\text{sum} = 3 + 3 = 6$

Iteration 4:  $\text{sum} = \text{sum} + i$

means,  $\text{sum} = 6 + 4 = 10$

Iteration 5:  $\text{sum} = \text{sum} + i$

means,  $\text{sum} = 10 + 5 = 15$

Iteration 6:  $\text{sum} = \text{sum} + i$

means,  $\text{sum} = 15 + 6 = 21$

Iteration 7:  $\text{sum} = \text{sum} + i$

means,  $\text{sum} = 21 + 7 = 28$

Iteration 8:  $\text{sum} = \text{sum} + i$

means,  $\text{sum} = 28 + 8 = 36$

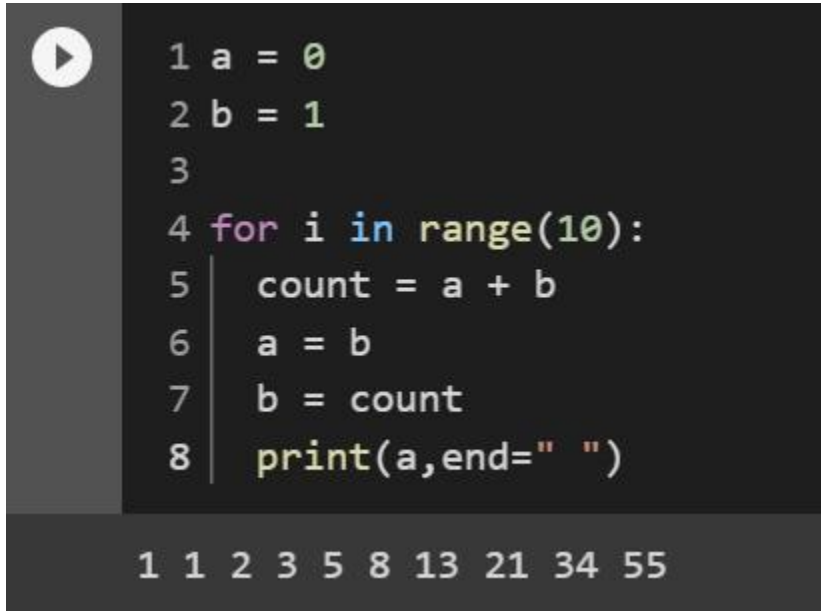
Iteration 9:  $\text{sum} = \text{sum} + i$

means,  $\text{sum} = 36 + 9 = 45$

Iteration 10:  $\text{sum} = \text{sum} + i$

means,  $45+10 = 55$  (Answer)

b) Print the first ten fibonacci numbers excluding 0.



```
1 a = 0
2 b = 1
3
4 for i in range(10):
5     count = a + b
6     a = b
7     b = count
8     print(a,end=" ")

1 1 2 3 5 8 13 21 34 55
```

Image 2.4.c: Solution for problem b.

### Algorithm visualization:

A line by line explanation is given below to understand the solution.

**Line 1 & Line 2:** Two variables are created. Inside the variables (a,b) 0 and 1 are stored respectively.

**Line 3:** Intentionally kept blank.

**Line 4:** A for loop is created. In the end parameter of the range() function is given 10 and other parameters are left empty. This will iterate the loop 10 times.

Additional information to clarify the concept: range(1,11), range(10,21), range(5,16) and every other valid arguments that can generate total 10 numbers would do the same thing.

**(Line 5 to line 8 will be processed inside the for loop)**

**Line 5:** The summation of stored value inside a and b ( $a+b$ ) is assigned to a new variable called count.

**Line 6:** The value of variable b in line 2 is now stored in variable a from line 1.

**Line 7:** The value of variable count from line 5 is now stored in variable b.

**(Inside The Loop)**

**Line 5 to line 8**

Iteration 1:  $\text{count} = a+b$

means,  $\text{count} = 0+1 = 1$

$a = b$

means,  $a = 1$

$b = \text{count}$

means,  $b = 1$

Iteration 2:  $\text{count} = a+b$

means,  $\text{count} = 1+1 = 2$

$a = b$

means,  $a = 1$

$b = \text{count}$

means,  $b = 2$

Iteration 3:  $\text{count} = a+b$

means,  $\text{count} = 1+2=3$

a = b

means, a = 2

b = count

means, b = 3

Iteration 4: count = a+b

means, count = 2+3=5

a = b

means, a = 3

b = count

means, b = 5

Iteration 5: count = a+b

means, count = 3+5 = 8

a = b

means, a = 5

b = count

means, b = 8

Iteration 6: count = a+b

means, count = 5+8 = 13

a = b

means, a = 8

b = count

means,  $b = 13$

Iteration 7:  $\text{count} = a + b$

means,  $\text{count} = 8 + 13 = 21$

$a = b$

means,  $a = 13$

$b = \text{count}$

means,  $b = 21$

Iteration 8:  $\text{count} = a + b$

means,  $\text{count} = 13 + 21 = 34$

$a = b$

means,  $a = 21$

$b = \text{count}$

means,  $b = 34$

Iteration 9:  $\text{count} = a + b$

means,  $\text{count} = 21 + 34 = 55$

$a = b$

means,  $a = 34$

$b = \text{count}$

means,  $b = 55$

Iteration 10:  $\text{count} = a + b$

means,  $\text{count} = 34 + 55 = 89$

a = b

means, a = 55

b = count

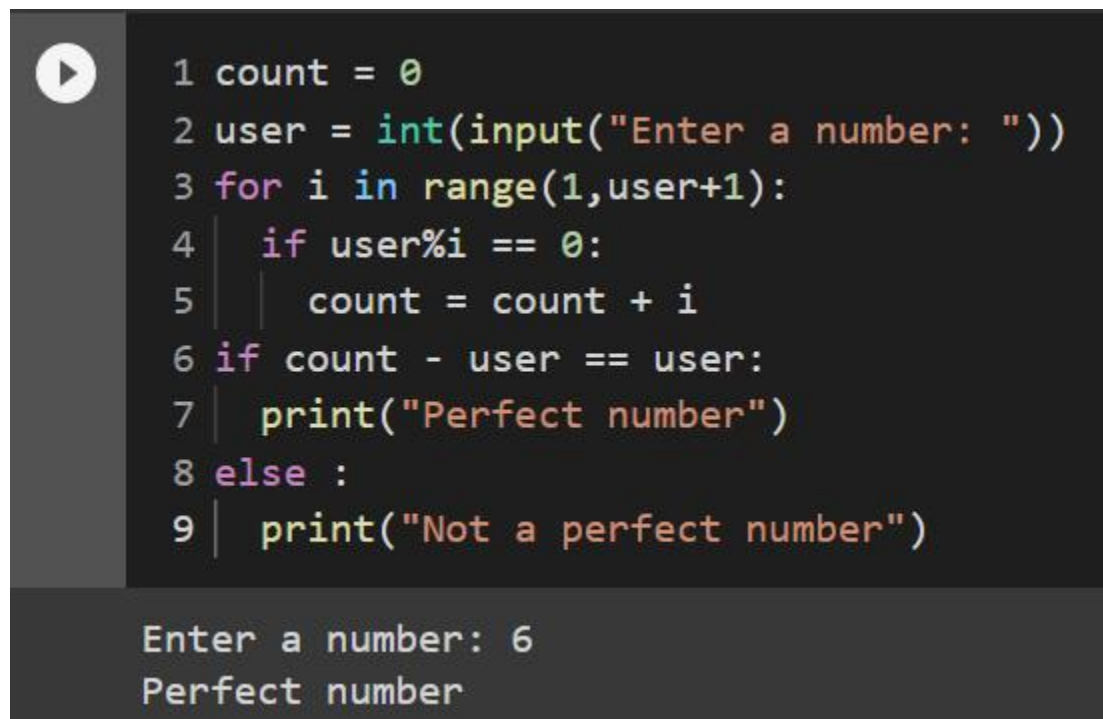
means, b = 89

Then the loop ends. So the entire loop yielded ten values of a and they

are 1,1,2,3,5,8,13,21,34,55. These are the first ten fibonacci numbers excluding 0.

c) Write a python program that reads a number from the user and prints “Perfect number” if that’s a perfect number. Otherwise, it will print not a perfect number.

**Concept:** In number theory, a **perfect number** is a positive integer that is equal to the sum of its positive divisors, excluding the number itself. For instance, 6 has divisors 1, 2 and 3 (excluding itself), and  $1 + 2 + 3 = 6$ , so 6 is a perfect number. (Source: Wikipedia)



```

1 count = 0
2 user = int(input("Enter a number: "))
3 for i in range(1,user+1):
4     if user%i == 0:
5         count = count + i
6 if count - user == user:
7     print("Perfect number")
8 else :
9     print("Not a perfect number")

```

Enter a number: 6  
Perfect number

Image 2.4.d: Solution for problem c

### Algorithm visualization

**Line 1:** A variable called “count” is created to use it later.

**Line 2:** The program takes a number from the user. (Let, the number be 6 for this problem.)

**Line 3:** A for loop is activated that will generate numbers ranged between 1 to the user’s input.

So the iteration process will generate six numbers ( $i = 1, 2, 3, 4, 5, 6$ ). During first iteration,  $i = 1$ , during second iteration,  $i = 2$  and the consistency will keep going till sixth iteration. Thus, the numbers will be generated.

### **(Inside The Loop)**

**Line 4:** If we divide the user input by the  $i$  numbers and the yielded remainders are 0 in every time the loop iterates then the valid  $i$  numbers (that satisfy the condition  $\text{user} \% i == 0$ ) will be summed up and stored in the count variable.

Iteration 1:  $i = 1$ ;  $\text{user} \% 1 = 0$  [ $\text{user} = 6$  is constant for every iteration]

Hence,  $\text{count} = \text{count} + i$

$= 0 + i = 0 + 1 = 1$  [initial value of count is 0. So,  $\text{count} = 0 + i$ ]

Iteration 2:  $i = 2$ ;  $\text{user} \% 2 = 0$ .

Hence,  $\text{count} = 1 + 2 = 3$

Iteration 3:  $i = 3$ ;  $\text{user} \% 3 = 0$ .

Hence,  $\text{count} = 3 + 3 = 6$

Iteration 4:  $i = 4$ ;  $\text{user} \% 4 = 2$ .

Hence, line 5 won’t be executed. The value for count is still 6.



Iteration 5:  $i = 5$ ;  $\text{user} \% 5 = 1$ .

Hence, line 5 won't be executed. The value for count is still 6.

Iteration 6:  $i = 6$ ;  $\text{user} \% 6 = 0$ .

Hence,  $\text{count} = 6 + 6 = 12$

### **The loop ends.**

The value for count stops at 12. Since we're checking whether user input is a perfect number or not, so we need to subtract the user input (6 here) from count's final value. Clarifying, check iteration 6. According to the definition of perfect numbers, perfect numbers are equal to its factor **excluding itself**. So we can't consider the 6th iteration process where  $i == 6$ . For a perfect number, if we subtract user input from count's final value and after the process, if the number is exactly equal to the user input then it will be a perfect number.

Count's final value = 12

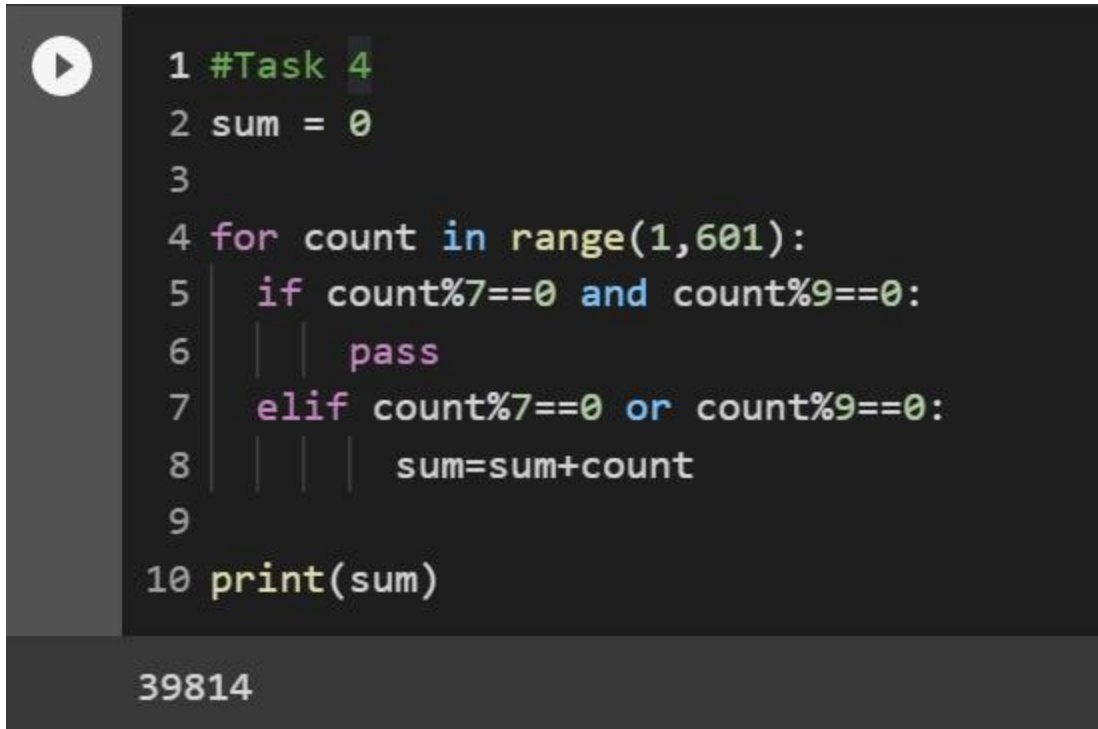
User input = 6

Determinant =  $12 - 6 = 6$ , which is equal to the user input.

So, 6 is a perfect number.

**Line 6 to 9:** Conditional statements are written to get the desired output.

d) Write a Python script of a program that adds all numbers that are multiples of either 7 or 9 but not both, up to 600 (including 600) i.e. 7, 9, 14, 18, 21..... and so on but not the numbers 63, 126, 189..... which are multiples of both 7 and 9.



```
1 #Task 4
2 sum = 0
3
4 for count in range(1,601):
5     if count%7==0 and count%9==0:
6         pass
7     elif count%7==0 or count%9==0:
8         sum=sum+count
9
10 print(sum)
```

39814

Image 2.4.e: Solution for problem d

### Algorithm visualization

**Line 2:** A variable 'sum' is created whose initial value is 0.

**Line 4:** A for loop is created and the number range has been selected by the user (1 to 600).

#### (Inside The Loop)

**Line 5:** Usage of conditional statement. If count's value (1,2,3,4...600) in range (from 1 to 600) is a multiple of 7 and 9 then...(continue from Line 6)

**Line 6:** **pass** keyword will pass the loop to next statements of the program. Hence, line 5 won't be executed and the program will start checking from line 7 to find out what to execute.

**Line 7:** If the boolean expressions of this line is True, then the program will try to execute the code under the if block which is written at line 8.

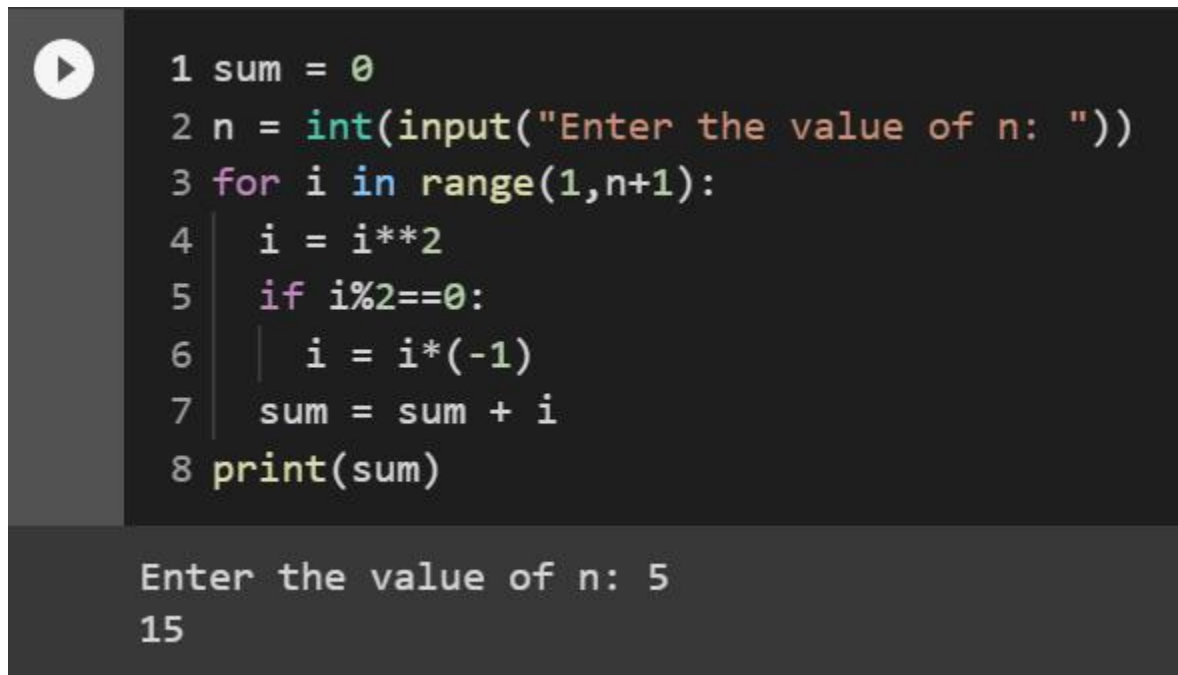
**Line 8:** The initial value of sum is 0. For every time the interpreter notices the conditional statements are true, it will increment sum's value according to the script.

The last iteration will yield the summed value of every numbers that are either multiple of 7 or 9 but not both. How syntax like "sum = sum + count" works in a loop we've seen the process in previous examples and algorithm visualizations.

**The loop ends.**

**Line 10:** print() function is used outside the loop so that it shows only the last value yielded by the iteration process, which is also the summed value.

e) Write a Python code that will calculate the value of y if the expression of y is as follows (n is the input):  $y = 1^2 - 2^2 + 3^2 - 4^2 + 5^2 \dots \dots \dots + n^2$



```

1 sum = 0
2 n = int(input("Enter the value of n: "))
3 for i in range(1,n+1):
4     i = i**2
5     if i%2==0:
6         i = i*(-1)
7     sum = sum + i
8 print(sum)

```

Enter the value of n: 5

15

Image 2.4.f: Solution for problem 5

## Algorithm visualization

**Line 1:** A variable 'sum' is created to store the values that might be yielded by future calculations.

**Line 2:** According to the question, the user gives an input to the program. The input is stored in the 'n' variable.

**Line 3:** A for loop is created.

### (Inside The Loop)

**Line 4:** Since, every numbers of the expression is in their squared form, so by writing  $i = i**2$ , we mention, numbers that are generated for every iteration of the for loop, the numbers should be squared. Suppose, the user inputted 5 as n's value. Then the generated numbers will be  $1^2, 2^2, 3^2, 4^2, 5^2$

**Line 5:** Notice the expression, before every even numbers the arithmetic operator is "-" So, need to write something and build the program in a way so that during the summation process the sign changes from + to - every time it operates summation of even numbers. Therefore, we wrote a conditional statement at line 5 and....(continue from line 6)

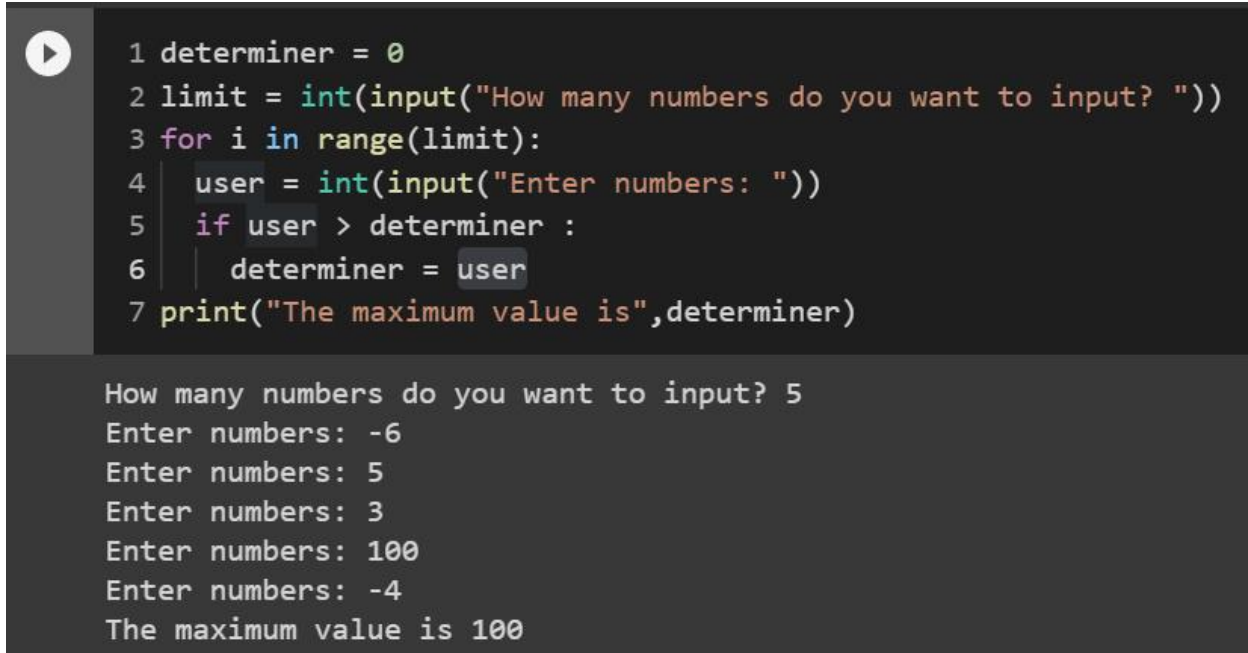
**Line 6:**  $i = i*(-1)$  we write,  $i = i*(-1)$  that changes the sign/turns the positive even number into its negative. Now the final expression formula is,  $y = 1^2 - 2^2 + 3^2 - 4^2 + 5^2 \dots\dots\dots + n^2$

**Line 7:** It adds every generated number according to the formula.

**The loop ends.**

**Line 8:** Prints the resultant.

f) Write a Python program that asks the user for a quantity, then takes that many numbers as input and prints the maximum of those numbers. (You can't use max(), min() built-in functions and list, also must input both positive and negative integers).



```
1 determiner = 0
2 limit = int(input("How many numbers do you want to input? "))
3 for i in range(limit):
4     user = int(input("Enter numbers: "))
5     if user > determiner :
6         determiner = user
7 print("The maximum value is",determiner)

How many numbers do you want to input? 5
Enter numbers: -6
Enter numbers: 5
Enter numbers: 3
Enter numbers: 100
Enter numbers: -4
The maximum value is 100
```

Image 2.5.g: Solution for problem f

### Algorithm Visualization

This problem is more difficult than the previous problems.

**Line 1:** A variable named “determiner” is created. This variable will be used later to determine the largest number.

**Line 2:** The program would ask the user to enter an integer with a prompt

“How many numbers do you want to input?” The input will be stored in a variable named “limit”. Suppose the user enters 5 as an input.

**Line 3:** A for loop is activated and the value of “limit” variable is passed to the range() function’s parameter. By doing so, the user can input as many numbers as he desires.

For this problem, the value is 5. That means the user can input 5 numbers to check which is the largest number among them.

**Line 4:** The user has to input 5 numbers.

**Line 5 and Line 6:** (May look confusing, focus carefully) The conditional statement

of line 5 is being read and will be executed under line 3's for loop. If the numbers

inputted by the user is greater than the determiner (initial value = 0) then in the

next line (line 6) the value of the "user" variable will be stored inside "determiner".

Clarifying, the previous value of the "user" variable (first inputted number) will be

the determiner for the second iteration. This time, the initial value of determiner will

be replaced by the first inputted number. The iteration will be continued until the

loop ends and at the end of the loop, we'll get the largest number.

Iteration 01: user = -6 which is not greater than determiner, 0 (unsatisfied condition).

Iteration 02: user = 5 which is greater than determiner, 0.

Iteration 03: Due to the satisfaction of iteration 02 conditional statement,

determiner = 5

user = 3 which is less than determiner 5 (unsatisfied condition)

Iteration 04: determiner = 5

user = 100 which is greater than determiner 5 (satisfied condition)

Iteration 05: determiner = 100

user = -4 which is less than determiner 5.

Iteration 06 isn't possible as the loop can only iterate 5 times according to the user's

desire. So, the last determiner's value is considered as the largest number among the five numbers. Therefore, the largest number is 100.

**Line 07:** `print()` function is used to show the largest number (determiner) on the console.

Note: This code won't work if all the inputs are negative integers.

## Exercises

**Objective 01:** Write the Python program that takes a number from the user and makes a multiplication table of that number. (1 to 10)

*Sample input and output(1)*

### Input

5

### Output

5x1=5

5x2=10

5x3=15

5x4=20

5x5=25

5x6=30

5x7=35

5x8=40

$$5 \times 9 = 45$$

$$5 \times 10 = 50$$

**Objective 02:** Write a Python program which reads two inputs (integers) from the user and prints the greatest common divisor (GCD) of the numbers.

**Objective 03:** Write a Python program which takes a number and prints the digits from the unit place, then the tenth, then hundredth, etc. (Right to Left). Do not use string indexing and consider the number as integer.

*Hint:* First to get the digit from the right side, we can take the remainder of the number using modulus (%) operator i.e. mod 10 to get the rightmost digit and print it. For dropping the last digit, we can perform floor division by 10 on the number and then continue the same to print the other digits as shown below.

$$32768 \% 10 = 8$$

$$32768 // 10 = 3276$$

Then,

$$3276 \% 10 = 6$$

$$3276 // 10 = 327$$

and so on

$$327 \% 10 = 7$$

$$327 // 10 = 32$$

$$32 \% 10 = 2$$

$$32 // 10 = 3$$

$$3 \% 10 = 3$$

$$3 // 10 = 0$$



Then you can stop the loop.

**Objective 04:** Write a Python code that will calculate the **value of x if the expression** of x is as given (n is the input):  $x = -1^3 + 2^3 - 3^3 + 4^3 - 5^3 + \dots + n^3$

*Sample input and output(1):*

**Input**

5

**Output**

-81

**Objective 05:** Write a python program that prints a square of size N using . where N will be given as input as illustrated in the samples.

*Sample input and output(1):*

**Input**

4

**Output**

....

....

....

....

*Hint:* You can use the concept of string concatenation or nested loops.

**Objective 06:** Write a Python program that asks the user for a quantity, then takes that many numbers as input and prints the maximum, minimum and average of those numbers. (The code should work for both negative and non-negative numbers)

*Sample input and output(1):*

**Input**

5

10

4

-1

-100

1

**Output**

Maximum 10

Minimum -100

Average is -17.2

**Objective 07:** Write a python program that takes two integers from the user and prints their factorial individually with a message shown in the samples and prints the summed value of the two factorials.

*Sample input and output(1):*

**Input**

Enter first number: 5

Enter second number: 4

**Output**

The factorial of 5 is 120

The factorial of 4 is 24

The summation of the two factorial is 144

**Objective 08:** Write a Python program which takes an input (integer number) from the user and display the summation of all the numbers which are multiples of 7 up to the user input.

*Sample input and output(1):*

### **Input**

50

### **Output**

196

**Objective 09:** Write a Python program that takes a number and display:

1. The Fibonacci numbers up to the input.
2. The summation of every Fibonacci number up to the input.

*Sample input and output(1):*

### **Input**

User input: 10

### **Output**

1 1 2 3 5 8

The summation of the Fibonacci numbers up to 10 is 20

*Sample input and output(2):*

### **Input**

User input: 100

**Output**

1 1 2 3 5 8 13 21 34 55 89

The summation of the Fibonacci numbers up to 100 is 232

**Objective 10:** Write a Python program using loop that can print numeric triangles as shown below:

1

22

333

4444

55555

**Objective 11:** Write a Python program using loop that can print numeric squares as shown below:

00000

11111

22222

33333

44444

**Objective 12:** Write a python program that prints a right-angled triangle of height N using incrementing numbers where N will be given as input.

*Sample input and output(1)*

**Input**

4

**Output**

1

12

123

1234

**Objective 13:** Write a Python program that takes a number consisting of multiple digits and displays the final product of each digit of that number.

*Sample input and output(1):*

**Input**

356

**Output**

90

*Sample input and output(2):*

**Input**

12349

**Output**

216

**Objective 14:** Write a Python script that reads a number from the user and displays whether the number is an Armstrong number or not.

*Sample input and output(1):*

**Input**

153

### Output

153 is an Armstrong number

*Sample input and output(2):*

### Input

120

### Output

120 is not an Armstrong number

Explanation: If every digit of a number is cubed and then summed, for an Armstrong number, the sum will always be equal to the given number.  $1^3 + 5^3 + 3^3 = 153$ . So, 153 is an Armstrong number. On the other sample input,  $1^3 + 2^3 + 0^3 = 9$  which is not equal to 120. So, 120 is not an Armstrong number.

**Objective 15:** A source code is scripted below. Find the output of the code.

```
user = int(input("Enter a number: "))
for i in range(1,user+1):
    for j in range(i,user+1):
        print(j,end="")
    print()
```

**Objective 16:** A source code is scripted below. Find the output of the code.

```
user = int(input("Enter a number: "))
for i in range(1,user+1):
    for j in range(1,i+1):
        print(j,end="")
    print()
```

## Chapter 03: Sequence Data Types

We already learned a little about the sequence data types in Python from chapter 1. String, list and tuple are the sequence data types. In this chapter, we'll discuss their characteristics, functions and usefulness in detail.

### 3.1 String

Strings are mainly a combination of alphabets and characters. Since computers can only process 0 and 1, strings are converted as a combination of 0 and 1 and we see them as what we input. In Python, strings are assigned into numbers based on the ASCII table and they represent Unicode characters. Strings are surrounded by single or double quotation marks. For example: 'Stealth-7' and "Dane9X." Every character we put inside a quotation mark becomes a string. Even whitespace is considered as a string if it's confined with quotation marks.

#### 3.1.1 String Attributes

**a) Conversion policy:** String is a kind of sequential data type. Unlike numerical data types, any type of numbers, alphabets, characters or their combination belong to the set of the strings. For example: 1 is an integer but when its written like '1' in Python, it becomes a string. Usually, every numerical data type can be converted into string but not every string can be converted into numerical data types. We can convert string '2' into an integer but we can not convert the string 'Roze' into an integer. (There's still a manual way to do so. By using ord() and chr() functions, string to integer and integer to string conversion is possible respectively.)

```
>>> user = 1
>>> string = str(user)
>>> print(string)
>>> print(type(string))
'1'
<class 'str'>
```

This program has no error and successfully converted an integer to an string. Now look at the following program:

```
>>> user = 'qwerty'
>>> integer = int(user)
>>> print(integer)
>>> print(type(integer))
```

When we run it, we'll see something like this:

```
ValueError: invalid literal for int() with base 10: 'qwerty'
```

These circumstances clarify that string to integer conversion and vice-versa is possible only when the object is a number.

**b) String on loops:** Strings can go through a loop. By doing so, we can attain specific characters and work with them when needed. An example is shown below.

```
>>> user = 'qwerty'
>>> for alphabets in user:
...     print(alphabets)
q
w
e
r
t
y
```

This program printed every alphabet of the word/string 'qwerty' in each new line.

**c) String indexing:** We can acquire a single character from a string using string indexing.

```
>>> # Print the first three characters of a string.
>>> var1 = 'BlueViking'
>>> character1 = var1[0]
>>> character2 = var1[1]
>>> character3 = var1[2]
>>> print(character1, character2, character3)
B l u
```



To index, we use “box brackets” [ ] with the variable where the string is stored. In the example shown above, `var1[0] = 'B'`, `var2[1] = 'l'` and `var1[2] = 'u'`.

Here, the 0th index is B, 1st index is l and 2nd index is u. Yes, you got it right. Indexing begins from 0. The last character of a string can be found at the string's (length - 1) index. Python has a built-in function named **len()** that returns the number of characters that exist in the string. For example, 'Apple' is a word whose length is 5. If the length of a string is 13, then the last character of the string is found at the 12th index.

The indexing process stated above is known as '**positive indexing**' where the indices are counted as positive numbers. Contrast process of this is called '**negative indexing**' where the indices are counted as negative numbers. Usually, negative indexing is used to reverse a string. The -1th index of a string is the last alphabet of the given string.

```
>>> var1 = 'Apple'
>>> last_c= var1[-1]
>>> print(last_c)
e
```

The first character of a string is found at -length index. Consider the same string 'Apple.' The length of this string is 5. So, the negative value of length is -5.

```
>>> var1 = 'Apple'
>>> var2 = var1[-5]
>>> print(var2)
A
```

If we write `var1[-len(var1)]` instead of `var1[-5]` it will show us the same output “A”.

**d) Length:** By using `len()` function, we can see how many characters are there in a string.

```
>>> #Example
>>> string = 'Zakaev may betray'
>>> length = len(string)
>>> print(length)
```

**Note:** Whitespaces are considered as string as well. Everything inside a ' ' is a string element in Python. For positive indexing cases, the highest index of a string is equal to the strings (length - 1). In contrast, the highest index of a string is equal to the negative value of length of the string (-length) for negative indexing cases.

We can also get the length of a string without using len() function. To do so, create a variable first and loop the string. For every iteration the loop will extract a character from the string. Keep counting these characters to get the length. The process is shown below.

```
>>> count = 0
>>> str1 = 'Happy'
>>> for i in str1:
...     count = count + 1
>>> print("The length of the string is",count)
```

Syntax written on the third line causes the program to generate 5 alphabets H,a,p,p,y respectively. Then all of these alphabets will go inside the loop. Every time a character goes inside the loop, the count's value increases by 1. At the final iteration, the count's value becomes 5. This is the length of the given string.

**e) Character searching:** A specific character/sequence in a string can be found by using 'in' keyword. If a character is not available in a string can be checked by using 'not in' keyword.

*Example 3.1:* Write a Python program to find if a can be found in the string "qwertysdghajgkd"

```
>>> user = "qwertysdghajgkd"
>>> if 'a' in user:
...     print("True")
>>> if 'qwerty' in user:
...     print("True")
>>> else :
...     print("False")
True
True
```

‘a’ and ‘qwerty’ both can be found in the string. So, the program displayed two True on the console.

*Example 3.2:* Check if ‘z’ can be found in "qwertysdghajgkd"

```
>>> user = 'qwertysdghajgkd'
>>> if 'z' not in user:
...     print("z is not in the given string")
>>> if 'z' in user:
...     print("z is found in the given string")
z is not in the given string
```

**f) String slicing:** We can slice a portion of a string and create a new string with the portion. Just specify the start index and end index separated by a colon. Step index does not need to be specified if we don’t want a pattern of cancellation while slicing the strings. The sliced string will consist of characters from the start index to the ending index’s previous character. Ending index’s character is excluded in the sliced/new string. The function is exactly similar to range() function. It has three parameters: start, end, step. The new string will include the character of the index given in the start parameter but will exclude the character of step parameter. By default, the jumping **step** from one index to its next index is 1.

*Example 3.3:* Extract the first three characters of a string.

```
>>> user = 'String'
>>> slice = user[0:3]
>>> print(slice)
Str
```

*Example 3.4:* A string ‘Ghost’ is given. Print the new string produced from the slicing of its 2nd to 5th index.

```
>>> user = 'Ghost'
>>> slice = user[2:5]
>>> print(slice)
ost
```

*Example 3.5:* A string 'Ivtisum' is given. Start slicing from 0th to 8th index with a gap of 2 steps and print the output.

```
>>> user = 'Ivtisum'
>>> slice = user[0:8:2]
>>> print(slice)
Itsm
```

If we do not specify any parameter, then there will be zero change to the existing string.

```
>>> string = 'Applause'
>>> new = string[:]
>>> print(new)
Applause
```

Since a string's last index is length - 1, so if we set length as the ending index then the string will be sliced from the specified beginning index to its last character.

```
>>> str1 = 'Henllo'
>>> str2 = str[0:len(str1)]
>>> print(str2)
Henllo
```

*Example 3.6:* A string is given. The string has both numbers and alphabets. Slice out the numbers and print the alphabets only.

```
>>> str2 = '123efghij'
>>> char = str2[3:len(str2)]
>>> print(char)
efghij
```

Keeping an argument empty (including step parameter) means to include every character to the end from the beginning (general slicing cases).

```
>>> user = 'asdfg'
>>> test = user[: :]
>>> print(test)
```

```
asdfg
```

Alternative code for the previous program:

```
>>> user = 'asdfg'
>>> test = user[len(user): :]
>>> print(test)
asdfg
```

Another alternative solution for the previous problem:

```
>>> user = 'asdfg'
>>> test = user[:len(user) :]
>>> print(test)
asdfg
```

By using the method of string slicing, we can also reverse a string. The process of reverse is known as negative indexing. In our previous examples, we were seeing the positive indexing where a string was being sliced from a lower index number to higher index number. To do a negative indexing, we need to slice the string from higher index number to lower index number. We must specify the **step** while doing negative indexing.

```
>>> user = 'STRING'
>>> reverse = user[6:0:-1]
>>> reverse2 = user[6:0:-2]
>>> print(reverse)
>>> print(reverse2)
GNIRTS
GIT
```

Usually, the process shown above is followed for general slicing cases. Keep the start and end arguments empty and as mentioned before, while reversing, we must denote the steps.

```
>>> user = 'asdfg'
>>> reverse = user[: : -1]
>>> print(reverse)
```

gfdsa

It's possible to reverse a string without using slicing too. If you still remember the concepts of loop then you may find it interesting.

```
>>> string = 'Etherea'
>>> begin_idx = len(string)-1
>>> for i in range(begin_idx,-1,-1):
...     print(string[i],end="")
```

Here, a string is taken. We determined the range() function's start argument as length of the string - 1. The end argument is -1. The step is -1 as well. So, the loop will begin generating the values of i from 7 (length of 'Etherea') and keep decrementing to -1. We can use print() function to print string[i]. As a result, when i = 7, the printed alphabet will be a. When i = 6, the printed alphabet will be e. When i = 5, the printed alphabet will be r. This will be consistent until i's value becomes -1. Since the end argument is -1 so the loop will be stopped at i = 0. On the 0th index of the string, we'll get the first alphabet 'E' of the given string. Thus, the whole string will be printed reversely.

**g) Unicode character casting:** Python has two built-in functions named ord() and chr(). These functions are opposite to each other. The function ord() takes a string argument of a single Unicode character and returns its integer value from the ASCII table. In contrast, chr() function reads an integer and returns a character conjugated with the integer on the ASCII table.

Examples of chr() function:

```
>>> alph = chr(97)
>>> print(alph)
'a'

>>> alph2 = chr(65)
>>> print(alph2)
'A'
```

Examples of ord() function:

```
>>> user = ord('a')
>>> print(user)
97
```

```
>>> user1 = ord('A')
>>> print(user1)
65
```

Notice that the ASCII value of 'a' is 97 and the ASCII value of 'A' is 65. Python is a case-sensitive programming language that can detect the uppercase-lowercase differences of a word or sentence. So by using the `chr()` and `ord()` functions, it's possible to convert a word into its uppercase or lowercase. The numeric difference between an uppercase and lowercase alphabet inside the `chr()` parameter is 32. Look attentively, when we passed 65 as an argument of `chr()` function it returned us 'A'. When we passed 97 as an argument, in that case the function returned 'a'. It denotes that if we add (+) 32 with an integer inside the `chr()` function that returns the uppercase alphabet of the existing alphabet (if it's actually an alphabet), then after adding 32 it will return the lowercase of the given alphabet. To convert a lowercase alphabet into uppercase, subtract 32. We can activate a loop over a given word/sentence in case it's needed to convert the entire word into its uppercase-lowercase or vice versa.

### Uppercase to lowercase:

```
>>> user = chr(78)
>>> print(user)           #Output : 'N'
>>> user2 = chr(78+32)
>>> print(user2)          #Output : 'n'
```

### Lowercase to uppercase:

```
>>> user = chr(112)
>>> print(user)           #Output: 'p'
>>> user2 = chr(112-32)
>>> print(user2)          #Output: 'P'
```

*Example 3.7:* Write the Python script of a program that reads a lowercase word from the user and returns it in uppercase.

```
>>> user = input("Enter a word: ")
>>> for i in range(len(user)):
...     upper_c = chr(ord(user[i])-32)
...     print(upper_c,end="")
```

To understand the code, let's hand simulate the third line. (Executing a script manually, by using pen and papers is known as hand simulation. It is also known as “tracing.” )

Suppose, the word given by the user is msi. The length of msi is 3. Hence, the generated values of i are 0,1,2 according to the for loop's given range. So, user[i] = m,s and i respectively and the loop will be iterated 3 times.

**Third line:** upper\_c = chr(ord(user[i])-32)

Iteration 01: user[i] = 'm'

$\text{ord}('m') - 32 = 109 - 32 = 77$

$\text{chr}(77) = 'M'$

Iteration 02: user[i] = 's'

$\text{ord}('s') - 32 = 115 - 32 = 83$

$\text{chr}(83) = 'S'$

Iteration 03: user[i] = 'i'

$\text{ord}('i') - 32 = 105 - 32 = 73$

$\text{chr}(73) = 'I'$

The loop ends. Now we can print the alphabets in a line by using end. The final output will be MSI.

*Example 3.8:* A string 'The Midnight' is given. Count the summation of the ASCII values of the string.

```
>>> sum = 0
>>> given = 'The Midnight'
>>> for i in given:
...     sum = sum + ord(i)
```



```
>>> print("The summation of the ASCII values is",sum)
The summation of the ASCII values is 1141
```

The explanation of this script is similar to the previous problem. When the for loop starts iterating, the ASCII values of each alphabet are added with the values of sum. The initial value of sum is 0. It updates per iteration.

**h) String concatenation:** “Concatenation” refers to merging things together. Several strings can be concatenated by using the “+” operator. When the “+” operator is used to concatenate strings, we call it an **overloaded operator**.

```
>>> var1 = 'abc'

>>> var2 = 'def'
>>> concatenate = var1 + var2
>>> print(concatenate)
abcdef
```

The same string concatenation can be done by using “\*” operator as well. In this case, “\*” is called a **repetition operator** instead of an **arithmetic operator**.

```
>>> user = 'rty'
>>> conc = user*3
>>> print(conc)
rtyrtyrty
```

String concatenation of non-empty and empty strings -

```
>>> var1 = 'qwerty'
>>> var2 = ' '
>>> var3 = var1 + var2
>>> print(var3)
qwerty
```

*Example 3.9:* A string ‘Fight for us’ is given. Make a new string with the vowels that exist in the string. Ignore the spaces in the new string.

```
>>> string = 'Fight for us'
>>> vowels = 'aeiou'
```

```

>>> new_string = ''
>>> for alph in string:
...     if alph in vowels:
...         new_string = new_string + alph
>>> print(new_string)
iou

```

**Explanation:** Lets hand simulate from line 4. “for alph in string” means for every element in the string. Line 5, “if alph in vowels”, this means, if the alphabet generated from the loop of line 4 is found inside vowels then the codes under that block should be executed (the program will go to line 6). So, the meaning of line 5 and line 6 is, for every element in the given string, if the elements are found in vowels then proceed to execute the codes under the if block. **This is a very important concept of string and loops and you’ll need to use this concept over and over again in this course.** Anyway, after the interpreter reaches line 7, it will add every vowel to the empty string (new\_string). By doing so, new\_string will be kept updated. When the iteration will be stopped, we’ll get every vowel that is found inside the given string.

The script written above, new\_string is a variable which carries an empty string. alph is the iteration variable. The hand simulation from line 4 to line 6 is described below.

Iteration 01: alph = ‘F’, which is not in vowels. So it won’t be concatenated with the empty string. The empty string won’t be updated.

Iteration 02: alph = ‘i’, which is in vowels. So, it will be concatenated with the empty string and the empty string will be updated.

new\_string = new\_string + alph = “” + ‘i’ = ‘i’

Iteration 03 to Iteration 07: alphas aren’t in vowels. There will be no change to the new\_string.

Iteration 08: alph = ‘o’, which is in vowels. So,

new\_string = new\_string + alph = ‘i’ + ‘o’ = ‘io’

Iteration 09 to Iteration 10: No change in new\_string as the alphabets aren’t vowels.

Iteration 10: alph = 'u', which is in vowels. So,

```
new_string = new_string + alph = 'io' + 'u' = 'iou'
```

Iteration 11: alph not in vowels. So, new\_string remains unchanged.

The iteration ends. 'iou' is the latest value of the new\_string. Now, look at the given string. Which vowels can you find there? Only 'iou', right? Yes.

**i) Mutability:** Strings are immutable. It's impossible to change the sequence of a string or to change an element/character of the string. By using **string indexing**, a certain object from a string can be attained but it cannot be changed anyhow **until we convert the string into a list**. A string can be forced to be mutable by doing so.

*Example 3.10:* Correcting the sentence 'My name Dane' to 'My name is Dane'

```
>>> string = 'My name Dane'
>>> b_list = list(string)

>>> b_list[7]= ' is '
>>> new_string = ''
>>> for i in b_list:
...     new_string = new_string + i
print(new_string)
```

My name is Dane.

**j) Multi-line string:** All of the strings we saw above were a single line string. In case if we need to write a paragraph, we can use the docstring method to write a multi line string. A triple quotation mark is used to create a docstring.

```
info = '''Anime is a curse to the
teenagers, children and adults. It's a mistake and
should be banned. '''
print(info)
```

Output

```
Anime is a curse to the
teenagers, children and adults. It's a mistake and
should be banned.
```

### 3.1.2 Program Analysis

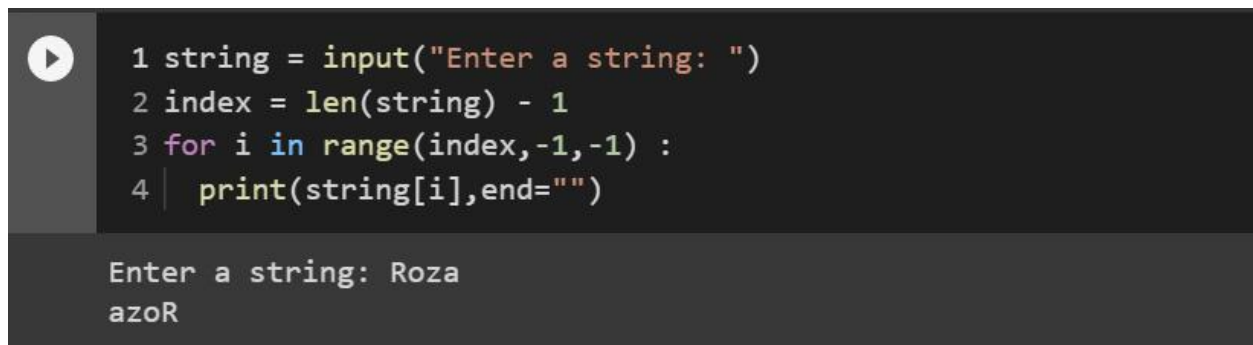
a) Write a Python program that takes a String as an input from the user and prints that String in reverse order without using string slicing and built-in reverse function.

*Sample input and output (1):*

Input: Python

Output: nohtyP

A sample solution for the given problem is given below.

A screenshot of a code editor with a dark background. On the left, there is a play button icon. The code is written in a light-colored font. It consists of four lines: 1. string = input("Enter a string: ") 2. index = len(string) - 1 3. for i in range(index, -1, -1) : 4. print(string[i], end="") Below the code, there is a text input field showing "Enter a string: Roza" and the output "azoR".

```
1 string = input("Enter a string: ")
2 index = len(string) - 1
3 for i in range(index, -1, -1) :
4     print(string[i], end="")

Enter a string: Roza
azoR
```

Image 3.1.2.a: Solution for problem a

#### Algorithm Visualization

**Line 1:** The program asked the user to input a string.

**Line 2:** A variable named “index” is made to store an integer which value is one digit less than the length of the given string.

**Line 3:** The range function is used taking the valid parameters to reverse the string while a for loop is also created. To the start, end and step parameters, the value of index, -1 and -1 will respectively be passed.

Suppose, the user inputted ‘Roza’ as the string, the length of the string is 4. So, the value

of “index” will be 3. According to the loop, the first value of the iteration variable (i) will be 3, then it will keep decrementing step -1 every time. Thus, the generated values for i comes out: 3,2,1,0.

**Line 4:** This statement has a print() function into which string[i] is written.

This determines, for every generated values of i due to the iteration process is an index of the given string and hence the characters that belong to each index will be printed sequentially abide by the for loop conditions written at line 4.

Iteration 1: i = 3, so, Output = string[3] = ‘Roza’[3] = a

Iteration 2: i = 2, so, Output = string[2] = ‘Roza’[2] = z

Iteration 3: i = 1, so, Output = string[1] = ‘Roza’[1] = o

Iteration 4: i = 0, so, Output = string[0] = ‘Roza’[0] = R

Output: azoR

In this way, the whole string is reversed. Note that, the print function is used inside the for loop so that it prints every character, not only the last character generated by the iteration. Also, end is used as a parameter of print() function so that each generated character doesn’t get printed in a newline. Rather, it prints sequentially every character in a line.

b) Write a Python program that will ask the user to input a string (containing exactly one word). Then the program should print subsequent substrings of the given string.

*Sample input and output (1):*

Input: Pacifia

Output: P

Pa

Pac

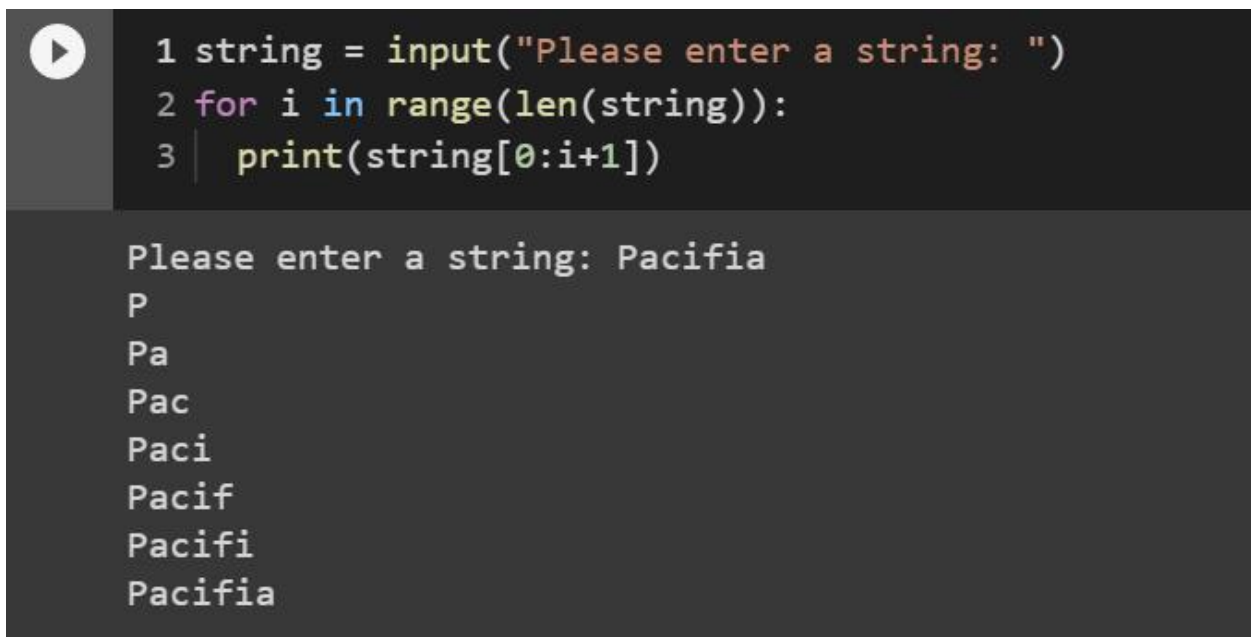
Paci

Pacif

Pacifi

Pacifia

A sample solution for the problem is given below.



```
1 string = input("Please enter a string: ")
2 for i in range(len(string)):
3     print(string[0:i+1])
```

Please enter a string: Pacifia

P

Pa

Pac

Paci

Pacif

Pacifi

Pacifia

Image 3.1.2.b: Solution for problem b

### Algorithm Visualization

**Line 1:** The user is asked to enter a string which will be stored inside a variable named “string”.

**Line 2:** A for loop is structured which will cause iteration according to the length of the given string which is placed as the argument of range() function. For example, if the length of the string is 3 then the loop will iterate 3 times.

**Line 3:** This is the most important statement. Read the problem again. We're told to print the subsequent substrings, so we are going to make sure the first character of the string remains unchanged every time the loop iterates. But the next characters should be updated per iteration. So, `string[0:i+1]` is written as the argument of `print()` function.

This will keep slicing the string in the following manner:

Let, the input given by the user to the "string" variable is 'Pacifia'

So, the length of the string is 7. As a result, the for loop will iterate 7 times. It will generate 7 values for `i` which will be the outputs.

Iteration 1: `i = 0`

Output = `string[0:0+1] = 'Pacifia'[0:1] = P`

Iteration 2: `i = 1`

Output = `string[0:1+1] = 'Pacifia'[0:2] = Pa`

Iteration 3: `i = 2`

Output = `string[0:2+1] = 'Pacifia'[0:3] = Pac`

Iteration 4: `i = 3`

Output = `string[0:3+1] = 'Pacifia'[0:4] = Paci`

Following this pattern, iteration 5,6 and 7 will respectively print Pacif, Pacifi and Pacifia in newlines.

c) Write a Python program that takes a string as an input from the user containing all small letters and then prints the next alphabet in sequence for each alphabet in the input. Consider 'a' as the next alphabet of 'z'.

*Sample input and output (1):*

Input: abcd

Output: bcde

*Sample input and output (2):*

Input: the cow

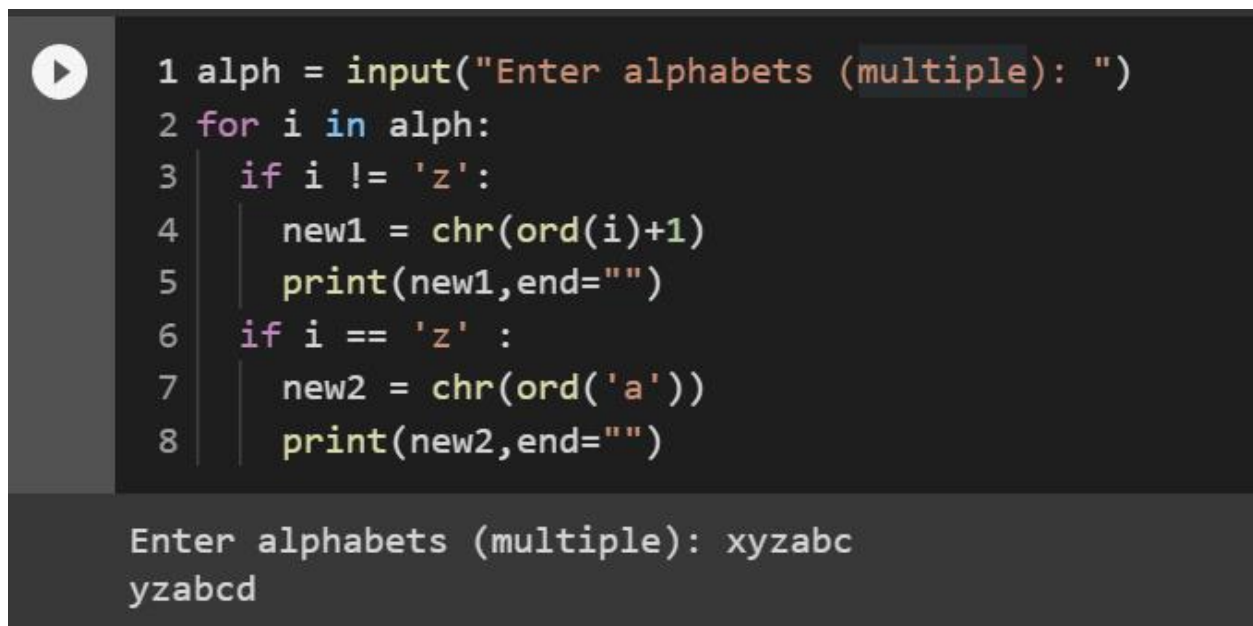
Output: uif!dpx

*Sample input and output(3):*

Input: xyzabc

output: yzabcd

A sample solution for this problem is given below:



```
1 alph = input("Enter alphabets (multiple): ")
2 for i in alph:
3     if i != 'z':
4         new1 = chr(ord(i)+1)
5         print(new1,end="")
6     if i == 'z' :
7         new2 = chr(ord('a'))
8         print(new2,end="")

Enter alphabets (multiple): xyzabc
yzabcd
```

Image 3.2.1.c: Solution for problem c

### Algorithm Visualization

**Line 1:** The program seeks for alphabets from the user to store those inside a variable named “alph”.



**Line 2:** Attains every alphabets from user input using a for loop. The alphabets are stored inside the iteration variable i/ i represents the alphabets.

**Line 3:** If the alphabet is not 'z' .. (continue from line 4)

**Line 4:** A new variable is created named new1. This will carry every values of i and every alphabets will be turned into its next alphabets as  $\text{chr}(\text{ord}(i)+1)$  is inside new1.

Let, i = 'a'

Then,  $\text{ord}('a')+1 = 97+1 = 98$

Now,  $\text{chr}(98) = 'b'$

So ultimately,  $\text{chr}(\text{ord}(i)+1) = b$

**Line 5:** print() function will print every generated alphabets for the condition written in line 3.

**Line 6:** The previous statements (line 3 to line 5) were valid if the alphabet is not 'z'.

From line 6, the program will check if 'z' belongs to the alphabets. If it belongs, then..

**Line 7:** A new variable named new2 will be created where z will be replaced by a.

The algorithm is similar to line 4's .

**Line 8:** Prints the output.

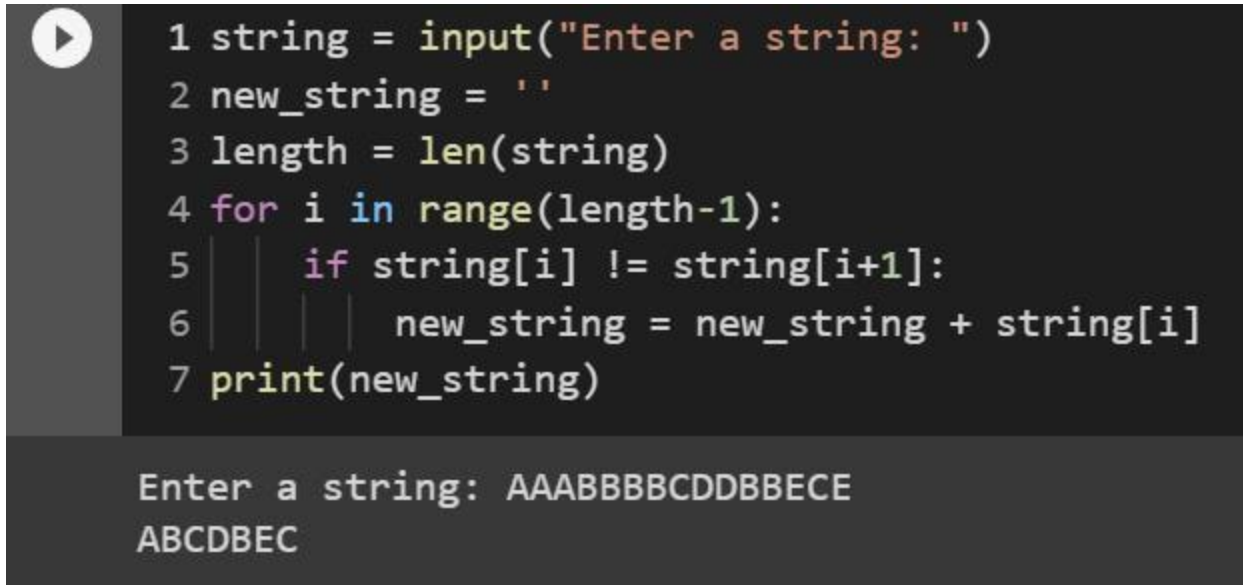
d) Given a string, create a new string with all the consecutive duplicates removed.

*Sample input and output (1):*

Input: AAABBBBCDDBBECE

Output: ABCDBECE

A sample solution is given below:



```

1 string = input("Enter a string: ")
2 new_string = ''
3 length = len(string)
4 for i in range(length-1):
5     if string[i] != string[i+1]:
6         new_string = new_string + string[i]
7 print(new_string)

```

Enter a string: AAABBBBCDDBBECE  
ABCDBEC

Image 3.1.2.d: Solution for problem d

### Algorithm visualization

**Line 1:** The program reads a string from the user.

**Line 2:** An empty string “new\_string” is created to use it later.

**Line 3:** The length of the given string is determined and stored in a variable named “length”

**Line 4:** The range is given length - 1. Otherwise, the loop will try to iterate even after when it already looped over every character of the string. This will cause an error.

**Line 5:** If the first index of the string is not similar to the second index then the alphabets will be moved to new\_string variable being concatenated with the empty string. Thus, consecutive duplicates will be removed.

Line 4 and Line 5 elaboration: What happens if we pass the value of the length as an argument of range() function?

Let, user given input is ABC

So, length of the string = 3

```
>>> for i in range(3):
...   if string[i] != string[i+1]:
```

The loop will iterate only 3 times. So at its first iteration, it will get two values of *i* together (two consecutive alphabets from the user's given input: AB). Then at its second iteration, the program will get the last alphabet of the string (C). Hence, the loop can't iterate for the third time but we commanded the loop to iterate thrice. As a result, we'll get an error message on the console showing the string index out of range.

**Tips:** Most of the problems related to string are usually solved by updating an empty list with newer values. So, try to clearly understand the concepts of “in”, “not in” keywords and when to create an empty string. If you have a good knowledge of Python strings and loops, be sure that you have already mastered 90% of the skill of problem solving that appears in this course. Best of luck.

## String Exercises

**Objective 01:** Write a Python code that reads a string from the user which is the mixture of alphabets and numbers and return the index of that number.

Sample Input & Output

Input	Output
qwer5tuy	4

**Objective 02:** Write the Python code that takes a string and displays the reversed string of the given string. (Try to solve it without using string slicing and by using string slicing both.)

Sample Input & Output

Input	Output
Mahjabeen Tamanna Abed	debA annamaT neebajhaM

**Objective 03:** Write a Python program to check whether the string is palindrome or not. You may need to convert every word into lowercase or uppercase after giving input. [**Palindrome Words:** A word is called a palindrome word if the original word and the reversed word are equal to each other. For example: noon, refer, civic, level etc.]

#### Sample Input & Output

Input	Output
Refer	The string is palindrome
121	The string is palindrome
qwert	The string is not palindrome

**Objective 04:** Write a Python script that takes two inputs from the user. One is a string and the other one is a number. The program should remove the string that is staying at the number's index.

#### Sample Input & Output

Input	Output
Etherea never loved Stealth 8	Etherea ever loved Stealth
Ghost 0	host

**Objective 05:** Write a Python program that reads a string from the user and counts its length ignoring the whitespaces.

Sample Input & Output

Input	Output
The Midnight	11
Dane 9X	6
Apple	5

**Objective 06:** Write a Python code that makes a new string with the even index characters from the given string. [Solve this problem by using string slicing and not by using string slicing]

Sample Input & Output

Input	Output
Swordfish	Sodih

**Objective 07:** Write a Python program that checks whether a digit exists inside a string.

Sample Input & Output

Input	Output
AvengedSevenfold-A7X	Digit exists

**Objective 08:** A string is given: 'anime is a mistake'. Write the script to convert every small letter into capital letters. [The ASCII range for smaller letters is 97-122 and capital letters 65-90]. Output is: ANIME IS A MISTAKE.

**Objective 09:** Write a Python program that takes a sentence from the user and extracts every vowel and a new string with the vowels. Also counts the number of vowels.

Sample Input & Output

Input	Output
-------	--------

Is Beluga a whale or YouTuber?	Ieuaaaeoooue  Vowels found: 5
--------------------------------	-------------------------------------

**Objective 10:** Write a Python program to delete every duplicate from a string. You may input the string to the program. Consider the alphabets as uppercase (if you wish to input any alphabet).

Sample Input & Output

Input	Output
AAEERTT65776JhLL	AERT657jhL

**Objective 11:** Write a Python code that takes input from the user and tells whether the input is Binary or not binary. You do not need to type cast the input string into integer. Consider the string a binary if the string only has 0 and 1s.

Sample Input & Output

Input	Output
0100111	Binary
010011w	Not binary

**Objective 12:** Write a Python program that will ask the user to input a string (containing exactly one word). Then print the ASCII code for each character in the String using the ord() function.

Sample Input & Output

Input	Output
Food	F: 70 o: 111 o: 111 d: 100

**Objective 13:** Write the Python code of a program that will take random consecutive duplicate letters from the user and create a new string with all the consecutive duplicates removed.

Sample Input & Output

Input	Output
AAABBBBCDDBBECE	ABCD BECE
AAaaaAaaAAaA	AaAaAa

**Objective 14:** Write a Python program that will take one input from the user made up of two strings separated by a comma and a space (see samples below). Then create a mixed string with alternative characters from each string. Any leftover characters will be appended at the end of the resulting string. [This problem is too difficult and large in size. Don't get depressed if you can't solve it. Take help from the solution.]

Sample Input & Output

Input	Output
ABCD,efgh	AeBfCgDh
ABCDENDFGH, ijkl	AiBjCkDIENDFGH

ijkl, ABCDENDFGH	iAjBkCIDENDFGH
------------------	----------------

**Objective 15:** Write a python program that takes 2 inputs from the user. The first input is a string and the second input is a letter. The program should remove all existence of the letter from the given string and print the output.

Sample Input & Output

Input	Output
Metallica Serbia a	Metllic Serbi

**Objective 16:** Write a python program that splits a given string on a given split character. The first input is a String and the second input is the character that will be used to split the first String.

Sample Input & Output

Input	Output
This-is-Dane	This is Dane
Etherea.never.loved.Stealth	Etherea never loved Stealth

**Objective 17:** A string is given: Python programming is very easy. Write the Python code that can display the following on the output console.

PyThOn PrOgRaMmInG iS vErY eAsY



**Objective 18:** Write a python program that takes 2 inputs from the user, where the first input is a string with length greater than 1. The second input is the index of the first given string from where you have to start reversing. After reversing the first input string from that index, print the new string back to the user. See samples below for clarification.

Sample Input & Output

Input	Output
72418 4	81427
12345 2	32145
aBcd1234defg 5	21dcBa34defg

**Objective 19:** Write a Python program that will ask the user to input a string (containing exactly one word). Then your program should print subsequent substrings of the given string as shown in the examples below.

Sample Input & Output

Input	Output
String	String tring ring ing ng g

**Objective 20:** Write a Python program that takes a string as an input from the user containing all digits and then adds +2 with every digit in the input and prints the output.

Sample Input & Output

Input	Output
111135	333357
000	222
11w	Invalid input

**Objective 21:** Write a Python script of a program that will read a word/sentence from the user and print the summation of the ASCII values of the alphabets.

Explanation: Assume, a word MJB. The ASCII values of the letters in this word are respectively 77, 74 and 66. The summation of these values is:  $77+74+66 = 217$ .

Sample Input & Output

Input	Output
Fear	382
Vengeance	908

## 3.2 List

Lists are multi-element data structures which are used to store multiple items in a single variable. In a list, items are stored inside a [ ] bracket. Each item of a list is separated by a comma. Various data types including integers, float, strings and even boolean expressions can be stored together inside a list. We create a list by putting items inside [ ]. For example,

```
a_list = [1,2,"apple", "banana",10>11]
```

### 3.2.1 List Attributes

**a) Conversion policy:** All kinds of built-in data structures and data types of Python can be placed inside a list as list items. A reminder, Python's built in data structures are **list**, **tuple**, **dictionary** and **set**.

**String to list:** For the type conversion between a string and list, confine the string with box brackets []. The class of the string will be converted into a list and the whole string will be an item of the list. But point to be noted, **the item's class remains the same, string. Just the whole data structure changes into a list.**

```
>>> user = 'String'
>>> a_list = [user]
>>> print(a_list)
>>> print(type(a_list))
>>> print(type(user))
['String']
<class 'list'>
<class 'str'>
```

By using the list() function, we can store every character of a string as a list item. The argument to be passed to the list() function are strings.

```
>>> user = 'String'
>>> a_list = list(user)
>>> print(a_list)
>>> print(type(a_list))
['S', 't', 'r', 'i', 'n', 'g']
<class 'list'>
```

**Integer to list:** The same formula to be followed as we did to convert string into a list.

```
>>> user = 1
>>> a_list = [user]
>>> print(a_list)
[1]
```

We used the **list()** function on string. But we cannot use it over integers.

```
>>> user = 12345
>>> a_list = list(user)
>>> print(a_list)
TypeError: 'int' object is not iterable.
```

To resolve the issue, we can convert the integer into a string.

```
>>> user = 12345
>>> a_list = list(str(user))
>>> print(a_list)
['1', '2', '3', '4', '5']
```

**b) Indexing:** Similar to string, list items can be indexed and specific items can be acquired. The method for indexing is exactly the same as string indexing. List indices counting begins from 0.

*Example 3.11:* Print the first name from the list.

```
>>> user = ['Alpha', 'Beta', 'Gamma', 'Sigma', 'Lambda']
>>> index = user[0]
>>> print(index)
Alpha
```

The last item of a list can be found at -1th index.

```
>>> user = ['Alpha', 'Beta', 'Gamma', 'Sigma', 'Lambda']
>>> print(user[-1])
Lambda
```

**c) Mutability:** Unlike strings, lists are mutable. We can add, change or replace items of a list. This can be done by using string indexing, insert() or append() function.

**Adding items to an existing list** by using append() function: *Example 1:* Given list = [1,2,3,4]. Add 5 to the list.

```
>>> given_list = [1,2,3,4]
>>> given_list.append(5)
>>> print(given_list)
```

```
[1,2,3,4,5]
```

**Warning:** `append()` function can only take one argument. So, we can't add more than one item to the list by using the `append()` function.

```
>>> given_list = [1,2,3,4]
>>> given_list.append(5,6)
>>> print(given_list)
TypeError: append() takes exactly one argument (2 given)
```

**Copying a list:** A list cannot be copied by doing variable assignment like we do to copy a string. When we have to copy a string, we usually create a new variable and store the existing string in this. The variables carrying the both strings aren't the same. If somehow a string is changed, the other string will still remain the same. But a list doesn't follow this. Since lists are mutable, the variable assignment method is not applicable. `[:]` can be used to copy a list. The built-in function **`copy()`** can also be used.

```
>>> a_list = [1,2,3,4]
>>> b_list = a_list[:]
>>> a_list.append(5)
>>> print(a_list)
>>> print(b_list)
[1,2,3,4,5]
[1,2,3,4]
```

See the example below to find out what happens if an existing list is copied by the following variable assignment method.

```
>>> a_list = [1,2,3,4]
>>> b_list = a_list
>>> a_list.append(5)
>>> print(a_list)
>>> print(b_list)
[1,2,3,4,5]
[1,2,3,4,5]
```

Here, `b_list` and `a_list` refer to the same list and if an element of `a_list` is changed, there will be a change in `b_list` too and vice-versa. Only the list shows this characteristic, string doesn't.

```
>>> string = '12345'
>>> store = string
>>> string = string + '1'
>>> print(string)
>>> print(store)
123451
12345
```

**Adding items to an existing list** by using **insert()** function: Insert function assists us to add a new item to a particular index **replacing** the existing item to its **next** index.

```
>>> given_list = ['Two', 'Three', 'Five', 'Six']
>>> given_list.insert(2, 'Four')
>>> print(given_list)
['Two', 'Three', 'Four', 'Five', 'Six']
```

Insert function has an exception. Usually the index -1 refers to the last element of a string/list or dictionary. So, by default we might think `insert(-1, 'item')` will add the item to the end of the list. This actually doesn't happen every time. Look at the example below.

```
>>> given = [1,2,3]
>>> given.insert(-1,4)
>>> print(given)
[1,2,3,4]
```

Now look at this.

```
>>> given = [1,2,3, 'String']
>>> given.insert(-1,4)
>>> print(given)
[1,2,3,4, 'String']
```

Here's another exception.

```
>>> given = [1,2,3,4, 'Tear', ['Lost, Breath'], 7.0]
>>> given.insert(-1, ['Apple'])
```

```
>>> print(given)
[1, 2, 3, 4, 'Tear', ['Lost, Breath'], ['Apple'], 7.0]
```

Insert function finds the data type sequence while adding items to an existing list. The insert() function has two parameters. One is for the index and the other is for the element. Suppose, the element we wish to add is a string type data and the existing list contains integer and string elements. In this case, insert() function will find out the sequence for string elements, if it successfully finds the sequence then our given element will be added to the last index of the sequence. If no string item is found in the list then the element is added to its length-1 index.

```
>>> given = [1,2,3]
>>> given.insert(-1,'str')
>>> print(given)
[1,2,3,'str',4]
```

Creating an empty list to update a list can be a good approach too. This manual process is an alternative to the insert() and append() function. This will be shown at **section f “Merging list.”**

**Changing items to an existing list :** When a certain item of a list needs to be changed (not replaced, rather to erase from the list), list indexing still can be used.

*Example 3.12:* Given list = ['Stealth','Dane','Zakaev','Etherea']. Now replace 'Zakaev' with 'Ghost'.

```
>>> given_list = ['Stealth','Dane','Zakaev','Etherea']
>>> given_list[2] = 'Ghost'
>>> print(given_list)
['Stealth','Dane','Ghost','Etherea']
```

Explanation: The program written above removes an existing item from a list by swapping a particular index's item with the desired item. In the line 2, we commanded in which index of the list 'Ghost' will be placed.

If it's necessary to change multiple items inside a list then we can make a range of change using the slice method.

*Example 3.13:* Given list = ['Price','Simon','Alex','Shepherd','Menendez','Adler']. Replace items from 3rd index to last index with 'Stealth', 'RayLe' and 'Dane'

```
>>> given_list =
['Price','Simon','Alex','Shepherd','Menendez','Adler']
>>> given_list[3:6] = 'Stealth','RayLe','Dane'
>>> print(given_list)
['Price','Simon','Alex','Stealth','RayLe','Dane']
```

Notice that, the **step** parameter here is empty. So by default the slicing process works for every item in the list. If we specify the step parameter then the slicing operation will maintain a removal sequence.

```
>>> given_list =
['Price','Simon','Alex','Shepherd','Menendez','Adler']
>>> given_list[0:6:2] = 'Stealth','RayLe','Dane'
>>> print(given_list)
['Stealth','Simon','RayLe','Shepherd','Dane','Adler']
```

Explanation: After mentioning :2 in the step parameter, the list items are replaced from first item to last item skipping each middle item between the two items.

**d) Length:** The number of items stored inside a list is defined as the length of the list. To know the length of a list, we can use **len()** function. The same function that we learned in string attributes.

```
>>> a_list = [1,2,4,3,5,6,9>8]
>>> print(len(a_list))
7
```

Length is an integer type value and determining length is useful when we need to loop over a whole list to get the indices and their items.

**e) List on loops:** A list can go through loops. The loop is able to extract the item that exists in a list.

```
>>> a_list = [5,6,7,'t', 2>3]
>>> for items in a_list:
```



```
>>> print(items)
5
6
7
t
False
```

**f) Merging lists:** Several lists can be merged and formed into a new list by using the “+” operator.

```
>>> a_list = [1,2,3,4]
>>> b_list = [5,6,7]
>>> c_list = [8,9,10]
>>> merged_list = a_list + b_list + c_list
>>> print(merged_list)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Merging lists by using the “+” operator is an alternative to the **append()** function.

```
>>> given = [1,2,3,4]
>>> new = given + [5]
>>> print(new)
[1,2,3,4,5]
```

While the **append()** function can only take one argument, as a result we can add only one item to the existing list, but by using the merging procedure we can add as many items as we want.

```
>>> given = [1,2,3,4]
>>> new = given + [5] + [6] + [7]
>>> print(new)
[1,2,3,4,5,6,7]
```

**g) Item searching:** To find out if a certain item is available in a list or not, membership operators “in”, “not in” are used collaboratively with the keyword “if”.

```
>>> a_list = ['R','S','45',45,40==30]
>>> if False in a_list:
...     print("True")
True
```

Explanation: The list shown above has 5 items. A boolean expression is also placed inside the list which is `40==30`. 40 is never equal to 30, so after the execution the resultant will come out as “False.”

```
>>> print(bool(40==30))
False
```

This means, the item “False” exists in the list. Hence, the program printed “True” by following the conditional statement scripted in the second line of the source code.

**h) List slicing:** By slicing a list creating an index range, we can create a new list with the items placed in the set range. List slicing can do the following things:

- Make a new list which is a subset of a list.
- Reverse the list items.
- Reverse list items following a sequence.
- Create a new list replacing items following a sequence.

**Making a sub-list of a list:** Specify the index from which the list item should be added and stopped.

```
>>> a_list = [1,2,3,4,5,6,7]
>>> b_list = a_list[2:7]
>>> print(b_list)
[3,4,5,6,7]
```

**Reversing list items:** The method is explained in the string slicing section under string attributes.

```
>>> a_list = ['Pen', 'Paper', 'Book', 'PDF']
>>> reverse = a_list[len(a_list):-1]
>>> print(reverse)
['PDF', 'Book', 'Paper', 'Pen']
```

**Reversing list items while following a sequence:** The method is explained in the string slicing section under string attributes.

```
>>> a_list = [1,2,3,4,5,6,7]
>>> reverse = a_list[: : -2]
```

```
>>> print(reverse)
[7,5,3,1]
```

**Creating a new list removing items while following a sequence:** Explained before.

*Example 3.14 :* Given list = ['Price','Simon','Alex','Shepherd','Menendez','Adler']. Replace items from 3rd index to last index with 'Stealth', 'RayLe' and 'Dane'

```
>>> given_list =
['Price','Simon','Alex','Shepherd','Menendez','Adler']
>>> given_list[3:6] = 'Stealth','RayLe','Dane'
>>> print(given_list)
['Price','Simon','Alex','Stealth','RayLe','Dane']
```

### 3.2.2 Program Analysis

a) Two lists are given. Print the common element between them.

**Given**

```
list1 = [1,2,43,23]
```

```
list2 = [2,4,35]
```

**Output**

2

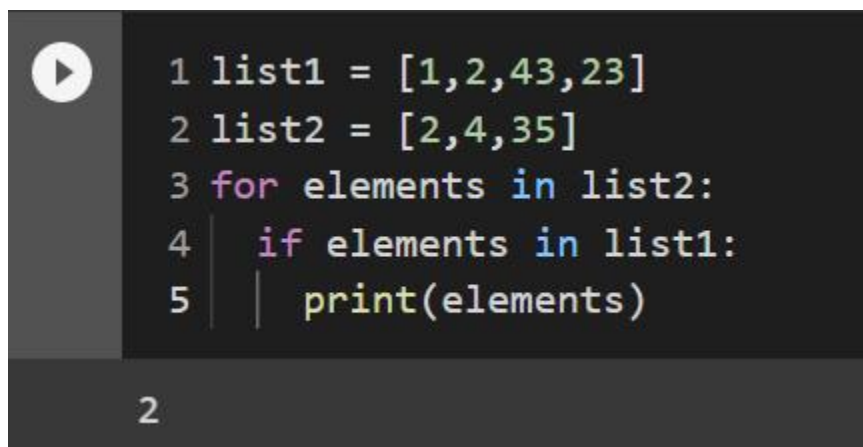


Image 3.2.2.a: Sample solution for 3.2.2.a

## Algorithm Visualization

**Line 1 & Line 2:** Two lists are given.

**Line 3:** A loop is going through the every item in list2 and thus the “elements” variable is storing the generated elements of list2. The elements are three integer numbers: 2,4 and 35.

**Line 4:** The list items which is stored inside the “elements” iteration variable will now check the list1 elements one by one to find whether a common element does exist between them or not. L(l)ist1 elements are 4 integers: 1,2,43 and 23.

**Line 5:** The print() function will display the common element(s) to the user. Since, 2 can be found in list1 and list2 both, so this list element/item will be displayed.

b) Write a Python program that takes list elements as an input from the user, then creates a new list excluding the first and last two elements of the given list and prints the new list. If there are not enough elements in the list to do the task, print “Not possible”.

Sample input and output(1):

### Input

10,20,30,40,50,60

### Output

‘30’,’40

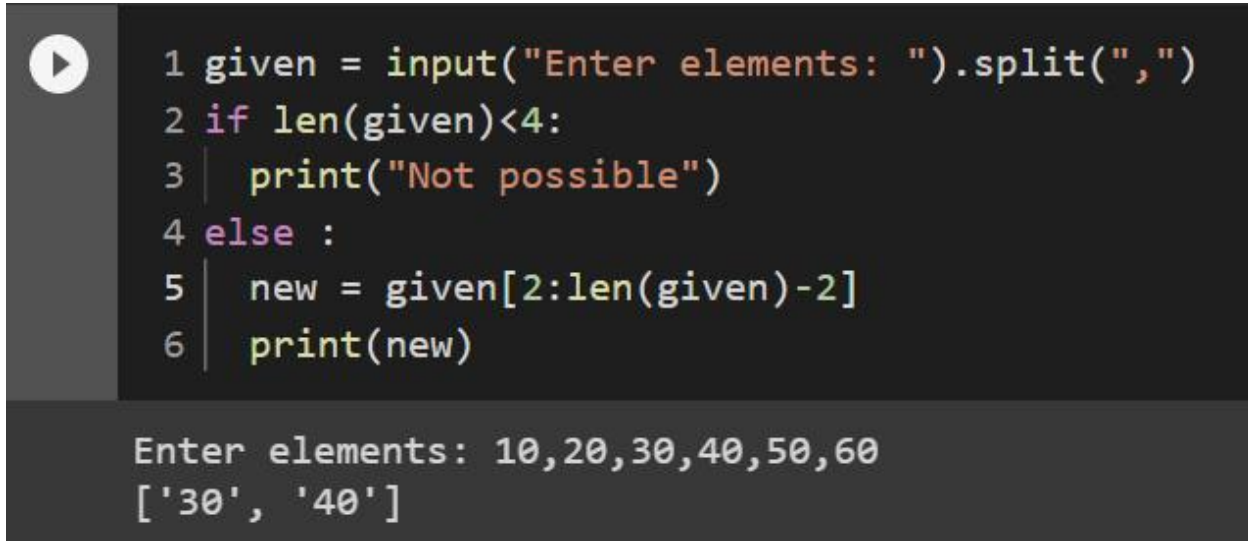
*Sample input and output(2):*

### Input

10,20,30

## Output

Not possible



```
1 given = input("Enter elements: ").split(",")
2 if len(given)<4:
3 |   print("Not possible")
4 else :
5 |   new = given[2:len(given)-2]
6 |   print(new)

Enter elements: 10,20,30,40,50,60
['30', '40']
```

Image 3.2.2.b: Sample solution for problem b

## Algorithm Visualization

**Line 1:** An user has to input list elements separated by a comma to the prompt. The `split()` function will then turn the elements into a list element and they'll be stored inside a list. Again, don't forget to specify a comma in the parameter of `split()` function.

**Line 2 and Line 3:** According to the problem, if the length is less than 4 then there will be not enough elements to make a list removing the first and last two elements. So, the program should print "Not possible"

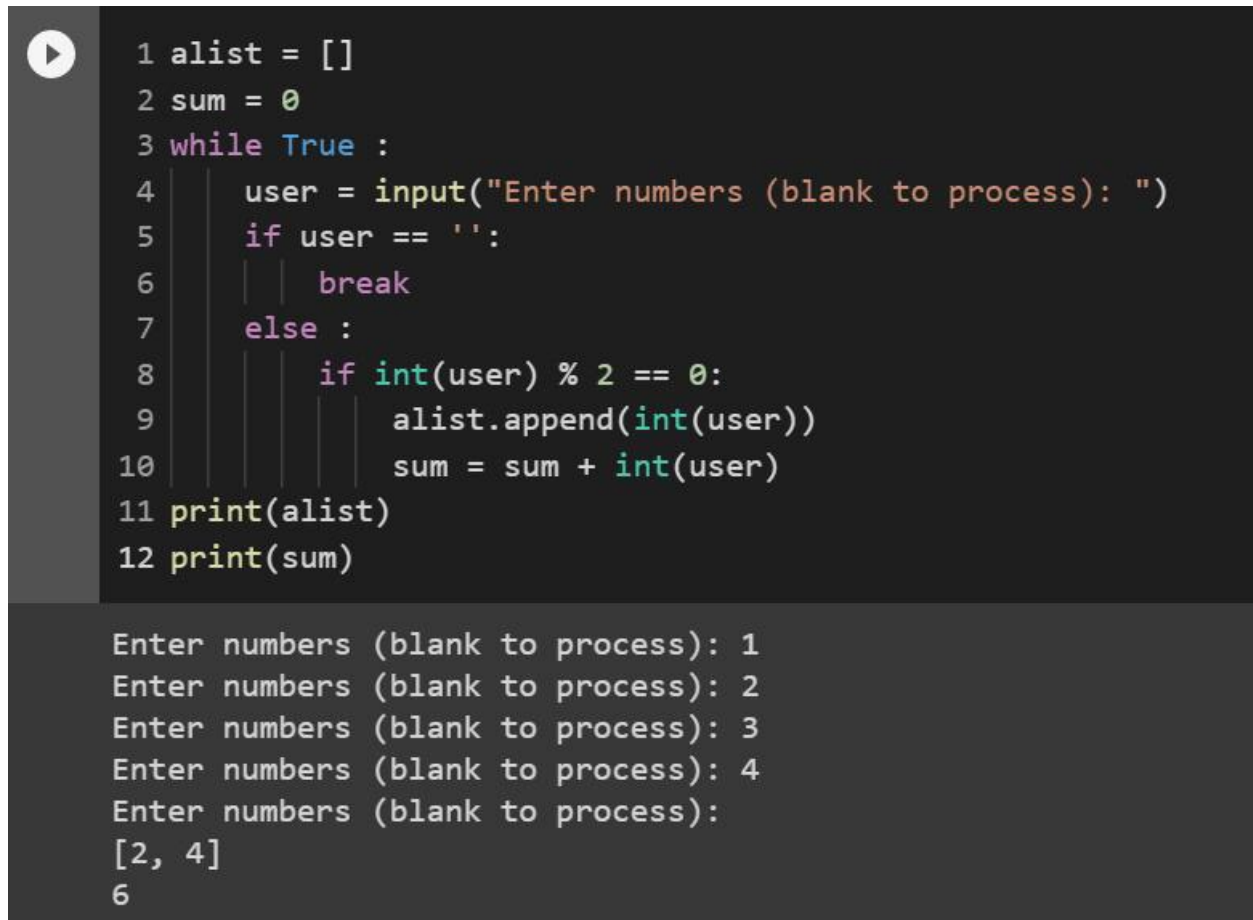
**Line 4:** An `else` condition is given. If the length of the list is more than 3, then there will be enough elements to make a list removing the last and first two elements.

**Line 5:** At this line, the syntax for the new list is written. We used list slicing method to remove the elements from 0 to 1th index and then elements from  $(\text{length}-2)$ th index.

The remaining indices and their elements will not be changed or removed.

**Line 6:** `print()` function will display the new list created from the previous line.

c) Write the Python code of a program that will take as many numbers as the user wants (if the user presses ENTER, the input prompt will stop appearing) and then make a list with the even numbers and display the sum of the (even) numbers.



```

1 alist = []
2 sum = 0
3 while True :
4     user = input("Enter numbers (blank to process): ")
5     if user == '':
6         break
7     else :
8         if int(user) % 2 == 0:
9             alist.append(int(user))
10            sum = sum + int(user)
11 print(alist)
12 print(sum)

```

Enter numbers (blank to process): 1  
 Enter numbers (blank to process): 2  
 Enter numbers (blank to process): 3  
 Enter numbers (blank to process): 4  
 Enter numbers (blank to process):  
 [2, 4]  
 6

Image 3.2.2.c: Sample solution for problem c

### Algorithm Visualization

**Line 1 and Line 2:** Two variables are assigned. One stores an empty list and another store 0. Every even number will be appended to the empty list “**alist**” and the summation of the numbers will be stored in “**sum**”.

**Line 3:** An infinity loop is created.

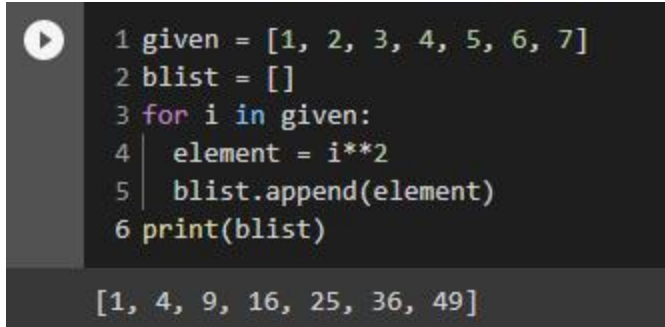
**Line 4:** The program lets the user input numbers (in **string** data type). Later, the input will be type casted into an integer.

**Line 5 and Line 6:** The program will keep giving an input prompt to the user until the user leaves a blank (only presses ENTER without giving a value). If the input is blank, the loop will be stopped by **break** statement and the interpreter will directly move to the end of the loop.

**Line 7 to Line 10:** If the input given by the user isn't a blank, then the program will continuously create input prompts for the user, where he will input numbers. A condition for even numbers is set in **line 8**. Numbers that will True the boolean statement will be appended to the list (**line 9**). The same numbers will be summed up at **line 10** since line 9 and line 10, both is written under the 8th line's conditional statement.

**Line 11 and Line 12:** These lines are outside the infinity loop. After the loop breaks, The program will finally print the list and the sum of the numbers.

d) Write a Python program that turns every item of a list into its square. Make changes in your list and see whether it works for other lists as well.



```

1 given = [1, 2, 3, 4, 5, 6, 7]
2 blist = []
3 for i in given:
4     element = i**2
5     blist.append(element)
6 print(blist)

[1, 4, 9, 16, 25, 36, 49]

```

Image 3.2.2.d Sample solution for problem d

### Algorithm Visualization

**Line 1:** A list is given.

**Line 2:** We created an empty list where the squared list items will be placed.

**Line 3:** A loop is going through the whole given list.

**Line 4:** “element = i\*\*2”. In line 3, we specified i to be the iteration variable. So, during the iteration, the given list elements will keep storing in i (i represents the items in every iteration). So, i = 1,2,3,4,5,6,7 and element = 1,4,9,16,25,36,48 (according to the variable assignment of **line 4**)

**Line 5:** Each iteration will generate a value that will be assigned to the element variable.

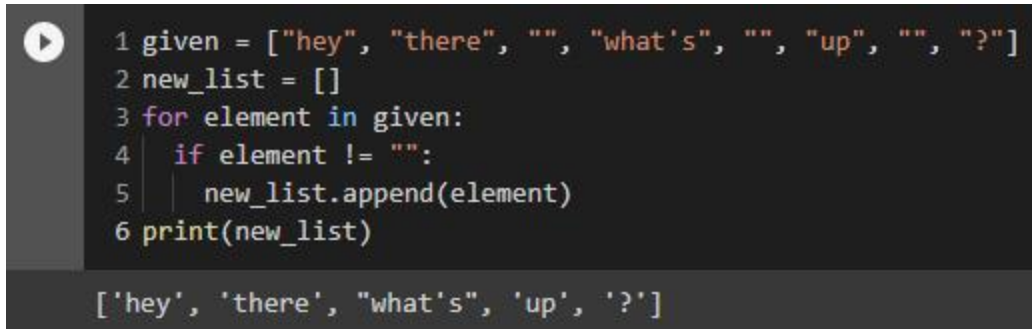
These values can be used to form a list if we pass the element variable to the argument of append() function. As the process will be done inside a for loop so values will be keep storing inside **element** variable and these will be appended to the blist.

**Line 6:** The list will be printed.

e) Write a Python program that removes all empty strings from a given list and prints the modified list.

Given list = [“hey”, “there”, “”, “what’s”, “”, “up”, “”, “?”]





```

1 given = ["hey", "there", "", "what's", "", "up", "", "?"]
2 new_list = []
3 for element in given:
4     if element != "":
5         new_list.append(element)
6 print(new_list)

['hey', 'there', "what's", 'up', '?']

```

Image 3.2.2.e: Sample solution for problem e

### Algorithm Visualization

**Line 1:** A list is given.

**Line 2:** An empty list is created named new\_list.

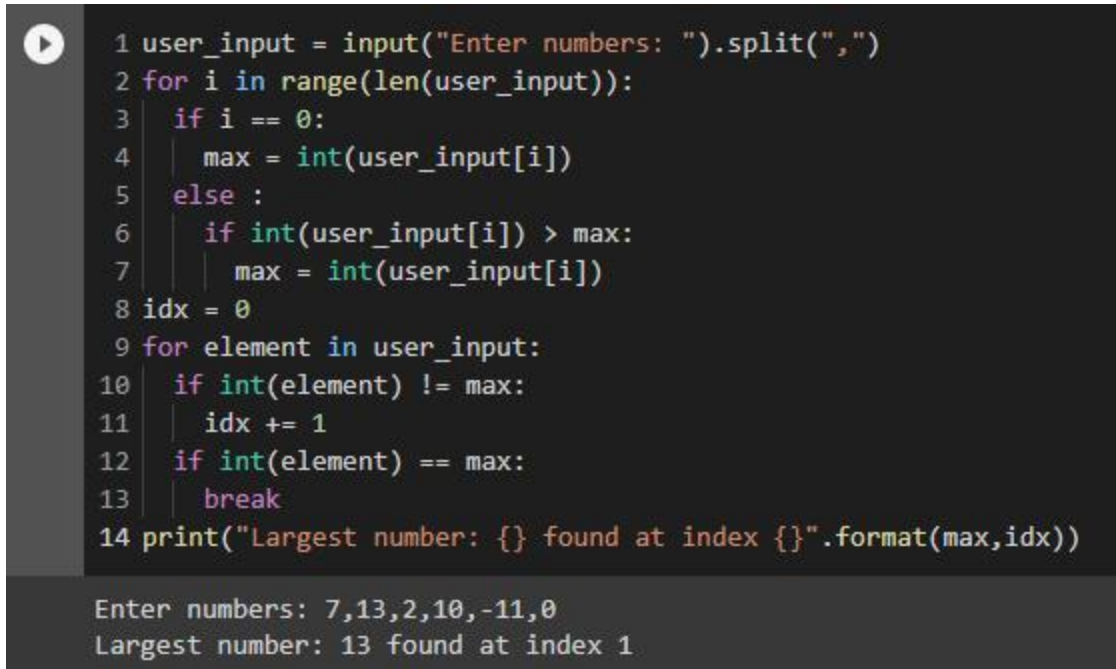
**Line 3:** Accessing every item of the given list. Accessed items are being stored inside iteration variable “**element**”.

**Line 4:** If items of the given list isn't "" (empty string) then.. (continue from line 5)

**Line 5:** Append the list items into the new\_list.

**Line 6:** Finally, the program prints the new\_list.

f) Write a Python program that reads a string containing 7 numbers separated by commas, then makes a list of those numbers and prints the largest number and its location or index position in the list. [Please don't use **max()**, **sort()** and **sorted()** functions]



```

1 user_input = input("Enter numbers: ").split(",")
2 for i in range(len(user_input)):
3     if i == 0:
4         max = int(user_input[i])
5     else :
6         if int(user_input[i]) > max:
7             max = int(user_input[i])
8 idx = 0
9 for element in user_input:
10    if int(element) != max:
11        idx += 1
12    if int(element) == max:
13        break
14 print("Largest number: {} found at index {}".format(max,idx))

Enter numbers: 7,13,2,10,-11,0
Largest number: 13 found at index 1

```

Image 3.2.2.f: Sample solution for problem f

### Algorithm Visualization

**Line 1:** The program takes input from users (integer numbers separated by comma) and turns them into a list items/stores them in a list using **split()** function.

**Line 2 to Line 7:** Let's assume the first element in the list to be the largest number and assign it to the **max** variable (Line 3-Line 4). As the loop continues, the value of **i** will keep increasing by +1 and access every element(integer numbers) of the list. If the program finds the current value of **max** is less than the present number of the list then the current value of **max** variable will be overwritten by the present value. Suppose, the list elements are [4,5,3]. So, the largest number according to the program at 0th index is 4 (we assume). At 1st index, the program checks whether number at this index is greater or less than the number of its previous index. 5 is greater than 3 so the current largest

number will be 5 and this will takeover the place of **max** variable replacing the previous value 3. But since 3 is not greater than 5, hence, even at 2nd index the largest number remains 5. In this way, the program will be able to find the largest number from the list.

**Line 8 to Line 13:** The largest number will only appear one time in the whole list. So, a loop is activated that will go through the entire list. A variable **idx** is assigned already. For each iteration, the value of **idx** increases by 1. When the list element finds the largest number, it will break the for loop and thus the value of **idx** will no longer increase. The last value of **idx** becomes the index of the largest number in the list.

**Line 14:** Prints the outputs.

## List Exercises

Complete the following objectives by writing Python programs. You do not need to input items if it's already given. In that case, you can change the given list and customize it in your own way to check your program and to make sure that your program works for every list related to the given lists. Try to match your outputs exactly with the shown sample outputs.

**Objective 01:** Count the sum and average of numbers from a list. Note, the list is storing multiple data types.

Sample Given & Output

Given	Output
['a',0,'b', 'Ron', 23,'c',15,'d',3, True ]	Sum : 41 Average: 10.25

**Objective 02:** Reverse items from a list. [Try to solve it without using slicing.]

Sample Input & Output

Input	Output
[100,200,300,400]	[400,300,200,100]

**Objective 03:** Write a Python program that multiplies all numbers in a list.

Sample Input & Output

Input	Output
[2,4,6,8,10,8,6,4]	737280

**Objective 04:** Find out the largest number from the given list.

Sample Given & Output

Given	Output
[-1,0,1,-2,2,-3,3,7,-7,5]	The largest number is 7

**Objective 05:** Find out the smallest number from a list.

Sample Given & Output

Given	Output
[-1,0,1,-2,2,-3,3,7,-7,5]	The smallest number is -7

**Objective 06:** Take an input from the user and then find the occurrence of that input in the given list.

Sample Input & Output

Input	Output
-------	--------

[1,2,3,3,3,3,4,5,6]	3 appears 4 times in the list
---------------------	-------------------------------

**Objective 07:** A range will be given by the user. Find the odd numbers between the range from the list and print their product. Given list = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]

Sample Given & Output

Input	Output
4 13	The new list is [5,7,9,11,13]

**Objective 08:** Create a list from user inputs and then make another new list removing the multiple words/items. Make sure the user can input every word within a prompt.

Sample Input & Output

Input	Output
Rain,Summer,Melancholy,Midnight,Summer,Night,Evening,Melancholy,Rain,Rain	['Rain', 'Summer', 'Melancholy', 'Midnight', 'Night', 'Evening']

**Objective 09:** Find the third largest number from a list and its location (index).

Sample Input & Output

Input	Output
[5,7,6,3,4,9,12,0,13,2,-1,-3]	Third largest number is 9 and found at 5th index

**Objective 10:** Find the positive integers from a list.

Sample Given & Output

Given	Output
['Ocean', 5.0, 'Helix', '7', '98', 98, 3>2, False, -2, 0, 4, 3-2, '3-1']	7 98 4 1

**Objective 11:** Find out the duplicate items from a list and print them.

Sample Given & Output

Given	Output
[25,35,45,[12,12,13],[12,12,13],45,1,1,8,8,7]	[[12,12,13]],45,1,8]
['Moonlight','Existence','Moonlight','Cloud','Leaves',Cloud']	['Moonlight', 'Cloud']

**Objective 12:** A list is given. Print the elements whose length is greater than 4.

Sample Given & Output

Input	Output
['Serene','Light','Rays','Sun','Breeze','Wind']	Serene Light Breeze

**Objective 13:** Take an integer input N from the user. The program will search for the elements which appear more than N times in the list. Then print the list of the elements. Given list = ['A','A','B','A','B','B','C','D',1,1,1,1]

Sample Input & Output

Input	Output
2	['A','B',1]

3	[1]
---	-----

**Objective 14:** Assume a list is given. The items of the list are names. A mistake occurred and now there are certain names that start with “ “. Put ‘x’ replacing the “ “ and print the final list.

Sample Given & Output

Input	Output
['Serena', ' ohan', 'Dave', 'Paul', ' en']	['Serena', 'Xohan', 'Dave', 'Paul', 'Xtherea']

**Objective 15:** Make a list by taking 4 numbers from the user. Try solving it without using the append() function.

Sample Input & Output

Input	Output
1 3 5 4	List: [1,3,5,4]

**Objective 16:** A list is given. Make a new list with the odd numbers from the existing list.

Sample given & Output

Given list	Output
Alist = [10,11,13,14,16,15,21,23,35]	[11,13,15,21,23,35]

**Objective 17:** Give 5 words to the program as input and print a list made with the words. Try solving it without using the split() function.

Sample Input & Output

Input	Output
Synesthesia is too scary	['Synesthesia', 'is', 'too', 'scary']

**Objective 18:** Pass a string input (numbers separated by comma and arbitrarily spaces) to the program and return the input as a list. You can use `split()` and `strip()` functions to finish this task.

Sample Input & Output

Input	Output
'1, 2, 3, 50, 4'	[1, 2, 3, 50, 4]

**Objective 19:** Two lists are given. Make a new list with the common elements from the two given lists and count the numbers of the common element.

Sample Given & Output

Given	Output
alist = [2,4,6,8,10] blist = [1,3,4,5,6]	List with common elements: [4,6] Commons found: 2

**Objective 20:** Make a program which will take as many numbers as an user wants to input, then when the user presses ENTER, a list will be created with the inputted items. The program will again ask for the users to input numbers like the previous way. Thus, another list will be created. Finally, merge the two lists and print the output.

Sample Input & Output

Input	Output
1 2 3	



4	
[Blank input/ENTER]	[1,2,3,4]
5	
6	
7	
[Blank input/ENTER]	[5,6,7]
	Merged list: [1,2,3,4,5,6,7]

**Objective 21:** Make a program which will take as many numbers as an user wants to input, then when the user presses ENTER, a list will be created with the inputted items. If the user presses R then already given inputs will be added into a list and the program will ask the user for further inputs to make another new list. In this way, the user can make as many lists as he wants. Finally, print all of the created list as another new list item. See the sample for clarification.

#### Sample Input & Output

Input	Output
1	
2	
3	
r	[1,2,3]
7	
6	
r	[7,6]
88	
99	
76	
ENTER	Final list: [[1,2,3],[7,6],[88,99,76]]

## 3.3 Tuples

Tuples are just like lists that can store different types of data inside a variable. The major difference between a tuple and a list is, a list is mutable but a tuple isn't. We can insert items into a list or order the items as we wish but we cannot do the same things when we're using tuples. Tuples are ordered and the collections inside the tuple are unchangeable/immutable. Another difference between tuples and lists is, tuples are written with first parentheses/ round brackets "()" where lists are formed with []. Even if we do not specify a (), by default Python will think of the elements as tuples (only if the elements are separated by a single comma). For example: 10,20,30 is a tuple in Python and so do (10,20,30). But while declaring a tuple with a single item we must specify a comma in order to make Python understand that its really a tuple, not an integer.

```
>>> tupl = 10,
>>> tupl2 = 10
>>> print(type(tupl))
>>> print(type(tupl2))
<class 'tuple'>
<class 'int'>
```

### 3.3.1 Tuple Attributes

**a) Conversion policy:** A tuple can be only converted into a list by using list() function. If we insert certain items into a () and separate them with a comma then the whole thing becomes a tuple. But the items' type inside the tuple remains unchanged.

```
>>> var1 = 'Anime'
>>> var2 = 'Mistake'
>>> var3 = 23
>>> var4 = 20+2>24
>>> a_tuple = (var1,var2,var3,var4) #Creating a tuple with variables
>>> a_list = list(a_tuple)          #Converting tuple into list
>>> print(type(a_tuple))           #Type checking of a_tuple
>>> print(type(a_list))            #Type checking of a_list
```

Output:

```

('Anime', 'Mistake', 23, False)
['Anime', 'Mistake', 23, False]
<class 'tuple'>
<class 'list'>

```

**b) Length :** To get the length of a tuple, `len()` function is used. We actually can use **`len()`** function to find out the length for every kind of sequence data type.

```

>>> a_tuple = (2,3,4,(5,6,7))
>>> print(len(a_tuple))
4

```

**c) Tuple indexing:** The method of tuple indexing is similar to list and string indexing.

```

>>> tupl = ('Metallica', 'Black Sabbath', 'Iron Maiden', 'Steelheart')
>>> index = tupl[2]
print(index)
Iron Maiden

```

**d) Tuple slicing :** Just like string and list slicing, the slicing procedure for tuple is the same.

```

>>> tupl = ('Metallica', 'Black Sabbath', 'Iron Maiden', 'Steelheart')
>>> slice = tupl[0:2]
print(slice)
('Metallica', 'Black Sabbath', 'Iron Maiden')

```

In our previous sections of ‘Sequence Data Types’ we noticed how strings and lists are sliced, negatively indexed, are sliced maintaining a pattern and reversed sliced maintaining a pattern. The exact methods are used for tuple slicing/reversing/negative indexing and slicing. So these attributes for tuples aren’t discussed in this book.

**e) Loop over a tuple:** A tuple can go through a loop like lists and strings.

```

>>> user = (1,2,3,4,5)
>>> for i in user:
...     print(i,end=",")
1,2,3,4,5,

```

**f) Mutability:** Tuples are immutable. It's not possible to change the items inside a tuple. But we still can force a tuple to be mutable by turning it into a list. The function `list()` can be used. After turning the tuple into a list, we can add/remove/replace items. When we're done, we just need to change the list into a tuple.

```
>>> a_tuple = (2,3,4,6)
>>> a_list = list(a_tuple)
>>> a_list.insert(3,5)
>>> a_tuple = tuple(a_list)
>>> print(a_tuple)
(2,3,4,5,6)
```

**g) Item questing:** Whether a random item is available in a tuple or not, “in”, “not in” keywords are used.

```
>>> team = ('Pacifia','Hellfire','X-Force','Alpha')
>>> if 'Hellfire' in team:
...     print("True")
True
```

**g) Tuple unpacking:** Every tuple usually stores random data types by packing themselves together inside parentheses. For example: `a_tup = ('Banana', 'Apple', 'Mango')`. By unpacking this tuple we can store and access each tuple item into different variables.

```
>>> a_tup = ('Banana', 'Apple', 'Mango')
>>> a,b,c = a_tup
>>> print(a)
>>> print(b)
>>> print(c)
Banana
Apple
Mango
```

### 3.3.2 Program Analysis

a) A tuple is given which informs about soldiers and their rank in Call of Duty: Mobile.

```
given_tuple = ([‘Stealth’,’Grandmaster’],[‘Masha’,’Legendary’],[‘Jester’,’Pro’])
```

Print the information in the following order. [Must follow the concept of **tuple unpacking**.]

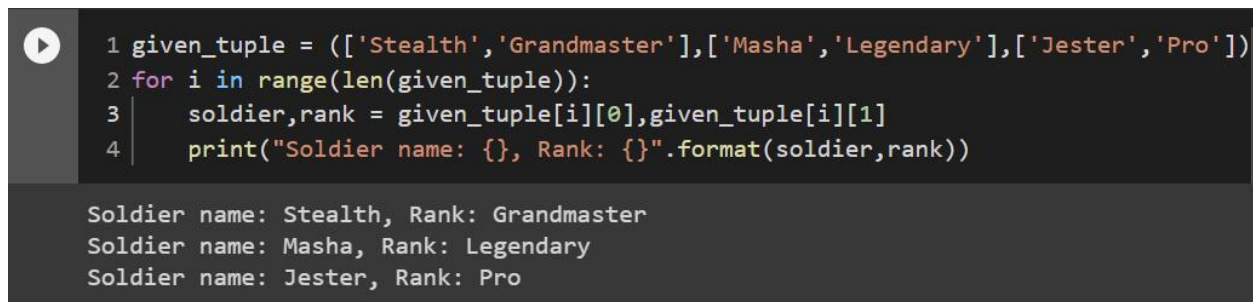
“Soldier name: [name of the soldier], Rank: [Soldier’s rank]”

#### Output

Soldier name: Stealth, Rank: Grandmaster

Soldier name: Masha, Rank: Legendary

Soldier name: Jester, Rank: Pro



```
1 given_tuple = ([‘Stealth’,’Grandmaster’],[‘Masha’,’Legendary’],[‘Jester’,’Pro’])
2 for i in range(len(given_tuple)):
3     soldier,rank = given_tuple[i][0],given_tuple[i][1]
4     print("Soldier name: {}, Rank: {}".format(soldier,rank))

Soldier name: Stealth, Rank: Grandmaster
Soldier name: Masha, Rank: Legendary
Soldier name: Jester, Rank: Pro
```

Image 3.3.2.a: Sample solution for problem a

#### Algorithm Visualization

**Line 1:** Given tuple is mentioned.

**Line 2:** A for loop is going through the tuple so that we can access the items. Currently,  $i = 0, 1, 2$  and hence,  $given\_tuple[i] = [‘Stealth’, ‘Grandmaster’], [‘Masha’, ‘Legendary’], [‘Jester’, ‘Pro’]$

**Line 3:** The given tuple is unpacked. Unpacked items now will be carried by two variables

named **soldier** and **rank**. This line will be executed under the for loop. The hand simulation of first iteration process and value update for the line is shown below.

Iteration 01: i = 0

```
given_tuple[i] = given_tuple[0] = ['Stealth', 'Grandmaster']
```

```
given_tuple[i][0] = given_tuple[0][0] = ['Stealth', 'Grandmaster'][0]
```

```
= 'Stealth'
```

```
given_tuple[i][1] = given_tuple[0][1] = ['Stealth', 'Grandmaster'][1]
```

```
= 'Grandmaster'
```

So, finally,

```
soldier,rank = ('Stealth', 'Grandmaster')
```

Similar to iteration 01, the values of i will keep incrementing and we'll get to know the name of every soldier and their ranks.

**Line 4:** The iteration will continuously unpack the tuple and store the values inside **soldier** and **rank** variable.

- Iteration 01 result: soldier,rank = 'Stealth', 'Grandmaster'
- Iteration 02 result: soldier,rank = 'Masha', 'Legendary'
- Iteration 03 result: soldier,rank = 'Jester', 'Pro'

Due to the print statement inside the for loop, the final outputs will appear like this:

Soldier name: Stealth, Rank: Grandmaster

Soldier name: Masha, Rank: Legendary

Soldier name: Jester, Rank: Pro

## Chapter 04: Dictionary

Python has only one built-in mapping data type and it is known as “**dictionary**.” A dictionary is a collection of **keys** and **values**, into which items are stored. Like **lists** and **tuples**, multiple data types can be stored in a dictionary. Dictionary shows similar attributes to **list** and **string** even though the data types don't seem to be resembling. List and strings are sequence data types and dictionary is a mapping data type. A dictionary is created by indicating the key,value pair separated by a “:” and confined into third brackets **{}**. Unlike tuples and string and being related to lists, dictionaries are mutable which means that a dictionary's item can be changed, updated or removed. One more major difference, list and tuples only store values but a dictionary stores keys and values together. The key,value pair feature is an advantage. This feature lets a user track data easier than other data types. Suppose a user needs to record the age and name of different persons. Using list and tuples won't be a good approach as the items are ordered, and have index features. So, the user has no short method to get a specific person's name/age if he doesn't know the index position of the person. But since the dictionary has key systems and the items can be accessed through the key, recording the data using the dictionary will be easier. The items can be accessed without knowing any index position. Additionally, the dictionary has no attribute like index position and so the dictionary is called the data type of unordered collections. The key,value pairs are here unordered and don't have any index position. A dictionary is given below which displays the names and ages of different persons.

```
dict1 = {'Rob':35 , 'Jones':25, 'Aron': 28}
```

Here, the first items (left items to the colon) are **keys** and the second items (right items to the colon) are **values**. These items can be accessed and updated.

### 4.1 Dictionary Attributes

**(a) Creating a dictionary:** A dictionary is created by using the curly braces where the keys and values are placed. For a single itemed dictionary the key and value is given separated by colon. This is the base structure of a dictionary. A dictionary without any key,value pair is known as an empty dictionary.

```
dict1 = {}
```



```
dict2 = {'A':3}
```

The first one is an empty dictionary and the second one is a single itemed dictionary. Item containing a key,value pair. To make a multiple itemed dictionary, we have to put a comma after each item and then place another key,value pair. Key,value pairs create the dictionary items.

```
dict2 = {'A':3, 'B':2}
```

**(b) Accessing items:** Every dictionary saves the object as **key** and **value** pairs. The values can be accessed from the keys. Take the previous dictionary as an example again.

```
dict1 = {'Rob':35 , 'Jones':25, 'Aron': 28}
```

If we want to print the age of Jones, then we can do so from the **key** 'Jones'.

```
>>> dict1 = {'Rob':35 , 'Jones':25, 'Aron': 28}
>>> print(dict1['Jones'])
25
```

**(c) Changing items:** To change an item, write down the existing key from the dictionary and assign a new value to it. Suppose, Rob from our previous example is now 36 years old. Now, we need to update his age in the dictionary. This is how we can do it:

```
>>> dict1 = {'Rob':35 , 'Jones':25, 'Aron': 28}
>>> dict1['Rob'] = 36
>>> print(dict1)
{'Rob':36 , 'Jones':25, 'Aron': 28}
```

**(d) Adding items:** Unlike list and strings, we cannot merge a dictionary using + operator. There's a function named **update()** that can add new items to an existing dictionary.

```
>>> dict1 = {'Rob':36 , 'Jones':25, 'Aron': 28}
>>> dict1.update({'Stealth': 23})
>>> print(dict1)
{'Rob': 36, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
```

Possible to add items without using the built-in function too.

```
>>> dict1 = {'Rob': 36, 'Jones': 25, 'Aron': 28}
>>> dict1['Stealth'] = 28
>>> print(dict1)
{'Rob': 36, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
```

**(e) Looping over a dictionary:** Like string, list and tuples dictionaries can go through a loop. By looping over a dictionary we can get keys, values or both (items) from a dictionary.

To access every key from a dictionary a for loop and **keys()** keyword is used.

```
>>> dict1 = {'Rob':36 , 'Jones':25, 'Aron': 28, 'Stealth': 23}
>>> for i in dict1.keys():
...     print(i)
Rob
Jones
Aron
Stealth
```

If you do not want to use the keyword, you can still get the keys. Look at the codes written below.

```
>>> dict1 = {'Rob': 35, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
>>> for i in dict1:
...     print(i)
Rob
Jones
Aron
Stealth
```

To access every value from a dictionary, a for loop and **values()** keyword is used.

```
>>> dict1 = {'Rob': 35, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
>>> dict2 = {}
>>> for i in dict1.values():
...     print(i)
36
```

```
25
28
23
```

These values can be accessed without using the built-in `values()` function too.

```
>>> dict1 = {'Rob': 35, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
>>> for i in dict1:
...     print(dict1[i])
35
25
28
23
```

To access every item from a dictionary, a for loop and **`items()`** keyword is used.

```
>>> dict1 = {'Rob': 35, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
>>> for i,j in dict1.items():
...     print(i,j)
Rob 35
Jones 25
Aron 28
Stealth 23
```

Without using the built-in **`items()`** function, we can proceed in this way.

```
dict1 = {'Rob': 35, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
for i in dict1:
    print(i,dict1[i])
```

Here's an alternative of the above program which returns the items as tuples:

```
>>> dict1 = {'Rob': 35, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
>>> for i in dict1.items():
...     print(i)
('Rob', 35)
('Jones', 25)
('Aron', 28)
('Stealth', 23)
```

**(f) Removing items:** The simplest way to remove an item from a dictionary is to use the `pop()` function. But since `pop()` is a built-in function, for the purpose of learning academic programming, I prefer to remove items manually instead of using `pop()` function. We can create an empty dictionary and exclude the item that we want to remove from the existing dictionary including the other existing items that we don't want to remove. A for loop is necessary to do this process.

### Removing a dictionary item without using pop() function

```
>>> dict1 = {'Rob': 35, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
>>> dict2 = {}
>>> for x,y in dict1.items():
...     if x != 'Rob':
...         dict2.update({x:y})
>>> print(dict2)
{'Jones': 25, 'Aron': 28, 'Stealth': 23}
```

### Removing a dictionary item by using pop() function

```
>>> dict1 = {'Rob': 35, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
>>> dict1.pop('Rob')
>>> print(dict1)
{'Jones': 25, 'Aron': 28, 'Stealth': 23}
```

Consider seeing 4th line and 2nd line respectively from the both scripts again. If we have commanded the program to find an item by its **value** then we would get an error. Python finds a dictionary item by its **key** item, not by the **value**. That's why we wrote "if x!= 'Rob'" in our first script and "dict1.pop('Rob')" in our second script.

**(g) Determining length of a dictionary:** By using `len()` function the length of a dictionary can be found. The length is actually the number of keys that exist in a dictionary.

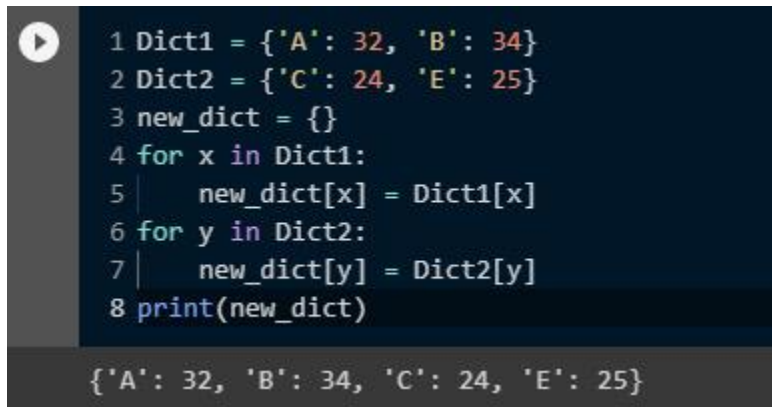
```
>>> dict1 = {'Rob': 35, 'Jones': 25, 'Aron': 28, 'Stealth': 23}
>>> print(len(dict1))
4
```

## 4.2 Program Analysis

(a) Two dictionaries are given. Make a new dictionary named `new_dict` with the items of the given two dictionaries.

`Dict1 = {'A': 32, 'B': 34}`

`Dict2 = {'C': 24, 'E': 25}`



```

1 Dict1 = {'A': 32, 'B': 34}
2 Dict2 = {'C': 24, 'E': 25}
3 new_dict = {}
4 for x in Dict1:
5     new_dict[x] = Dict1[x]
6 for y in Dict2:
7     new_dict[y] = Dict2[y]
8 print(new_dict)

```

`{'A': 32, 'B': 34, 'C': 24, 'E': 25}`

Image 4.2.a: Solution for problem a

### Algorithm Visualization

**Line 01 & Line 02:** Two dictionaries are given.

**Line 03:** A new empty dictionary is created naming it `new_dict`, according to the question.

**Line 04:** A for loop is run, to access every key of `Dict1`. By running the loop, current values of `x`:

`'A', 'B'`.

**Line 05:** Here we wrote, `new_dict[x] = Dict1[x]`. Assignment of this syntax will be done following the table shown below.

Values of x	<code>new_dict[x]</code>	<code>Dict1[x]</code>	<code>new_dict</code>
A	<code>new_dict['A']</code>	32	<code>{'A':32}</code>
B	<code>new_dict['B']</code>	34	<code>{'A':32,'B':34}</code>

Since the loop of line 4 runs by mentioning `'for x in Dict1'`, so when we write `Dict1[x]` in line 5, `Dict1[x]` represents every value of `Dict1`. But there's no for loop running over `new_dict`.

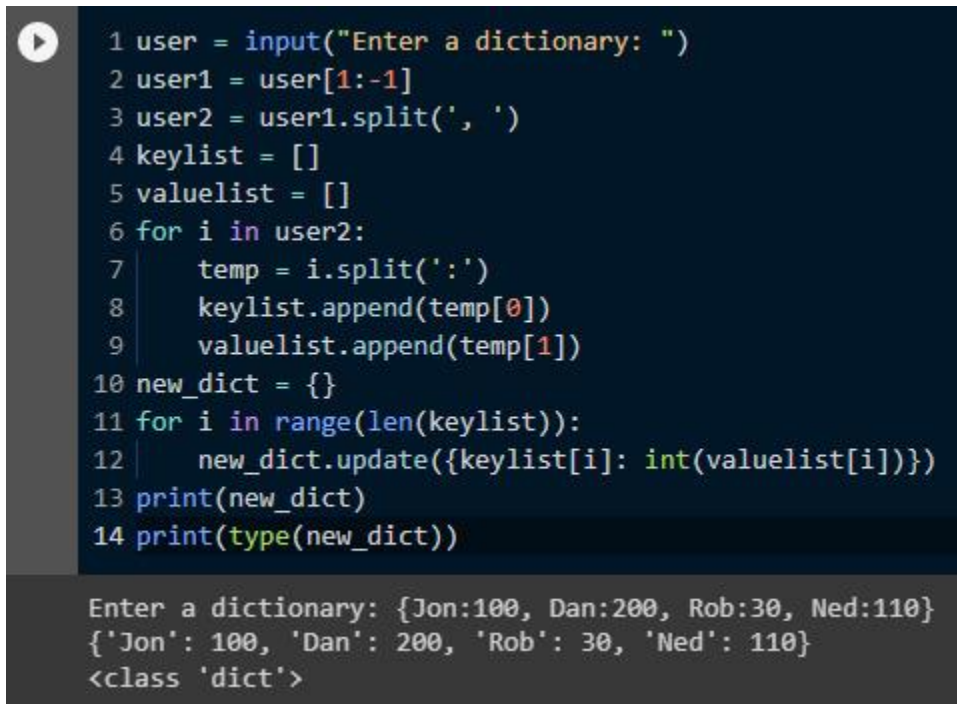
Hence, the assignment of the line is : `new_dict['A'] = 32`. The same syntax to add items to a dictionary. As a result, the empty dictionary will be updated with `{'A':32}`

**Line 06:** A for loop is run on `dict2` to access the keys.

**Line 07:** Same process of line 5.

**Line 08:** Prints the output.

(b) Write a Python program that will take a dictionary input as string type and return the dictionary as dictionary type. The values of the dictionary will be numbers and the keys can be words or letters. For example, `user_input = {'A':30}`. Output `{'A':30}`  
The type of the input is 'str' and the type of the output is 'dict'



```

1 user = input("Enter a dictionary: ")
2 user1 = user[1:-1]
3 user2 = user1.split(', ')
4 keylist = []
5 valuelist = []
6 for i in user2:
7     temp = i.split(':')
8     keylist.append(temp[0])
9     valuelist.append(temp[1])
10 new_dict = {}
11 for i in range(len(keylist)):
12     new_dict.update({keylist[i]: int(valuelist[i])})
13 print(new_dict)
14 print(type(new_dict))

```

Enter a dictionary: {Jon:100, Dan:200, Rob:30, Ned:110}  
{'Jon': 100, 'Dan': 200, 'Rob': 30, 'Ned': 110}  
<class 'dict'>

Image 4.2.b: Solution for problem b

## Algorithm Visualization

**Line 01:** The user will give input to the program.

**Line 02:** A dictionary starts with { and ends with }. So, it's necessary to remove these parentheses.

**Line 03:** On the basis of a comma, separated items will be stored as list items in user2 variable. For example, the dictionary sent to the program as input is, {'Jon':100, 'Dan': 200, 'Rob': 30, 'Ned':110}. So, the items of user2 right now is: ['Jon:100', 'Dan:200', 'Rob:30', 'Ned:110']

**Line 4 & Line 5:** Two variables are assigned to store the keys and values of new\_dict.

**Line 06:** A for loop is run over user2.

**Line 07 to Line 09:** By hand simulating the source code, the following results are found.

**[Inside the for loop]**

i	temp	keylist (temp[0])	valuelist (temp[1])
'Jon:100'	['Jon','100']	'Jon'	100
'Dan:200'	['Dan','200']	'Dan'	200
'Rob:30'	['Rob','30']	'Rob'	30
'Ned:110'	['Ned','110']	'Ned'	110

Therefore, the whole for loop will update the values of keylist and valuelist. The final result of this for loop:

keylist = ['Jon', 'Dan', 'Rob', 'Ned']

valuelist = ['100', '200', '30', '110']

**Line 10:** An empty dictionary named new\_dict is assigned.

**Line 11:** Another for loop is run. This time the loop has range() function included. The range is the length of keylist. For this example, the length of keylist is 4. (We could use the length of valuelist instead of keylist as well.)

**Line 12:** The execution process of line 12 is shown below using a trace table.

**[Inside the for loop]**

i	keylist[i]	valuelist[i]	new_dict {Keylist[i] : int(valuelist[i])}
0	'Jon'	'100'	{'Jon':100}
1	'Dan'	'200'	{'Jon':100, 'Dan':200}
2	'Rob'	'30'	{'Jon':100, 'Dan':200,'Rob':30}
3	'Ned'	'110'	{'Jon':100, 'Dan':200,'Rob':30, 'Ned':110}

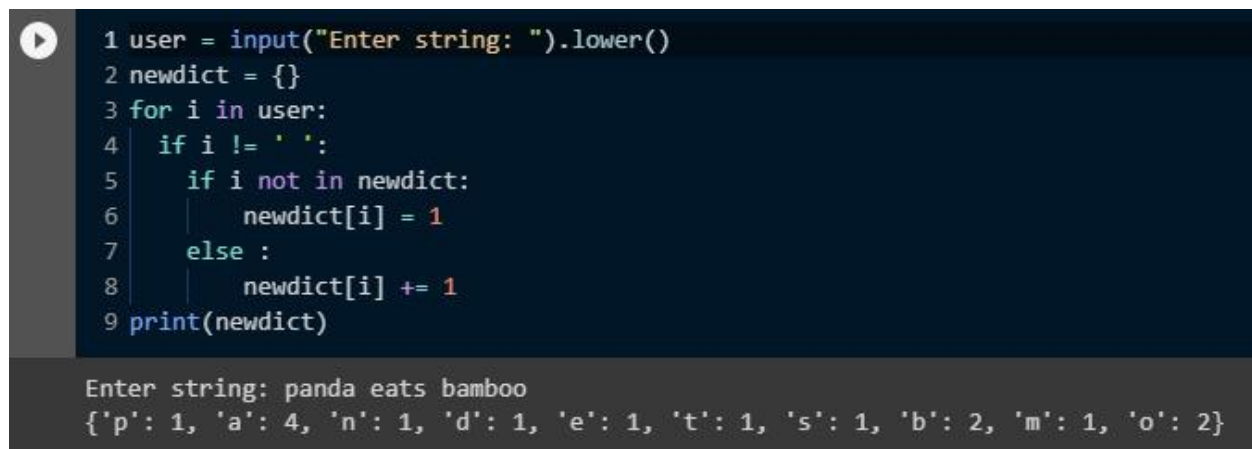
**Line 13:** Finally, print() function is used to print the dictionary (new\_dict).

**Line 14:** We can check the data type too. See the console, the type is dict.

(c) Write a Python program to count the frequency of each character of a user inputted sentence. [No case-sensitivity is necessary.]

Sample input: Panda eats bamboo

Output: {'p':1, 'a':4, 'n':1, 'd':1, 'e':1, 't':1, 's':1, 'b':2, 'm':1, 'o':2}



```

1 user = input("Enter string: ").lower()
2 newdict = {}
3 for i in user:
4     if i != ' ':
5         if i not in newdict:
6             newdict[i] = 1
7         else :
8             newdict[i] += 1
9 print(newdict)

```

Enter string: panda eats bamboo

{'p': 1, 'a': 4, 'n': 1, 'd': 1, 'e': 1, 't': 1, 's': 1, 'b': 2, 'm': 1, 'o': 2}

Image 4.2.c: Solution for problem c



## Algorithm Visualization

**Line 01:** An input prompt to let the user input a sentence, which will be turned into lowercase to neglect case-sensitivity issues.

**Line 02:** An empty dictionary is assigned to store the final items. The name of the dictionary is `newdict`.

**Line 03:** A for loop is run over the user.

**Line 04 to Line 08:** The operations inside the for loop are depicted below via a trace table. The summary of these source codes are: if the program finds a space in the user input then it will be ignored. Without spaces, other characters will be added to the dictionary as keys and initially their values will be 1. If the same key appears again in the input, then the values of the keys' will be kept incremented by +1.

### [Inside the for loop]

i	new_dict[i]	new_dict
p	p	{'p':1}
a	a	{'p':1,'a':1}
n	n	{'p':1,'a':1,'n':1}
d	d	{'p':1,'a':1,'n':1,'d':1}
a	a	{'p':1,'a':2,'n':1,'d':1}

In this manner the program will check the whole sentence and update the character's frequency and characters in `new_dict`.

**Line 09:** Prints the dictionary.

**Note:** Line 07's syntax under the else condition may look spiny to some people.

`new_dict[i] += 1` and `new_dict[i] = new_dict[i]+1` denotes the syntax with the same meaning.

On the second iteration of the program, `i = a`. So, according to the line 05's if condition,

`new_dict['a'] = 1`

This means, `{'a':1}`

Then, at the 5th iteration, the value of `i = a` again. Now, the program will go to the else statement of line 07.

Previously, `{'a':1}`

Currently,

`new_dict[i] += 1` [The syntax under else block]

> `new_dict['a'] += 1`

> `new_dict['a'] = new_dict['a'] + 1`

According to Python's variable assignment theory, if we imagine the syntax as a mathematical equation, then the left hand side of the equation stated above stays unchanged and functions like a variable in which values are stored. But the right hand side's `new_dict['a']` works like a value. The value here is 1 (from the first iteration).

As a result, `new_dict['a'] = 1 + 1`

> `new_dict['a'] = 2`

So, finally the dictionary becomes `{'a':2}`

(d) Write a Python program that finds the largest value with its key from a given dictionary.

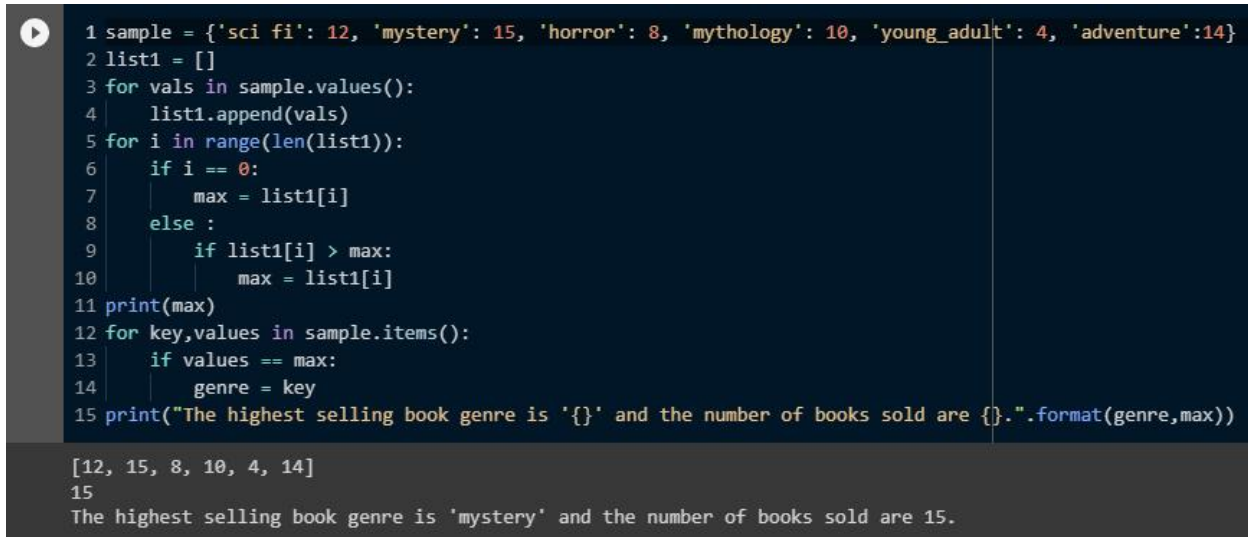
Try to solve it without using built-in `max()` function.

**Given:**

`{'sci fi': 12, 'mystery': 15, 'horror': 8, 'mythology': 10, 'young_adult': 4, 'adventure':14}`

**Output:**

The highest selling book genre is 'mystery' and the number of books sold are 15.



```

1 sample = {'sci fi': 12, 'mystery': 15, 'horror': 8, 'mythology': 10, 'young_adult': 4, 'adventure':14}
2 list1 = []
3 for vals in sample.values():
4     list1.append(vals)
5 for i in range(len(list1)):
6     if i == 0:
7         max = list1[i]
8     else :
9         if list1[i] > max:
10            max = list1[i]
11 print(max)
12 for key,values in sample.items():
13     if values == max:
14         genre = key
15 print("The highest selling book genre is '{}' and the number of books sold are {}".format(genre,max))

```

[12, 15, 8, 10, 4, 14]

15

The highest selling book genre is 'mystery' and the number of books sold are 15.

Image 4.2.d: Solution for problem d

## Algorithm Visualization

**Line 01:** A sample dictionary is given. We need to work with this.

**Line 02:** An empty list is assigned named list2 (Here we'll store the values of the given dictionary)

**Line 03 to Line 04:** The values of the dictionary are stored into list2.

**Line 05 to Line 10:** A for loop is run on list2 to find out the maximum number from the list. In our previous chapters, we already saw how this for loop extracts the maximum/minimum number from a list.

**Line 11:** This is an unnecessary line. The maximum value is printed to check if our previous codes were right.

**Line 12:** Another for loop is run to access the items (key,value pair) of the dictionary.

**Line 13:** For every iteration the keys and values of the dictionary will be stored into the two iteration variable **key** and **values**.

**Line 13 to Line 14:** We already found the maximum number. That moment when the value of **values** variable and the value of **max** variable will be the same, a new variable called **genre** will be created and it will store the corresponding key of **values**. For example, at the second iteration of this for loop, the program will find 15 as a value of the dictionary. 15 is also the value of the max variable. In this way, the program successfully finds the largest value from the dictionary.

**Line 15:** Prints the result on the console.

## 4.3 Exercises

Write Python programs to do the following objectives.

**Objective 01:** A user will give a sentence/string input to the user. Your task is to make a dictionary where the keys will be vowels and consonants and values will be how many times vowels and consonants are found in the sentence.

Sample Input & Output

Input	Output
Argentina won the FIFA world cup 2022.	{'Vowels': 10, 'Consonants': 17}
Celestia	{'Vowels': 4, 'Consonants': 4}

**Objective 02:** A dictionary is given to you, where a person's ID and occupation is given. Make a program that takes the ID number from a user and prints the person's information.

```
dict1 = {'Alex':(9999,'Student'), 'Drake':(5835,'Soldier'), 'Nishi':(3450,'Student'),
'Rowan':(4325,'Jobholder')}
```

Sample Input & Output

Input	Output
5835	{'Alex': Soldier}

3450	{'Nishi': Student}
------	--------------------

**Objective 03:** Assume, a dictionary is given to you. Find the key with the lowest value.

Sample Given & Output

Given	Output
my_dict = {'A':10, 'B': 5, 'C':12, 'D': 4}	D
my_dict = {'X': 4, 'Y': 7, 'Z': 3}	Z

**Objective 04:** Write a Python function named extractor that will take a string/sentence as an argument and prints-

- How many vowels, consonants, digits and special characters are there (ignoring the whitespaces.)
- Find out the ASCII average of even index characters and odd index characters of the sentence and make a dictionary named even\_dict and odd\_dict respectively where the key will be the beginning alphabet of the first even index character (for even\_dict) and the last odd index character (for odd\_dict) . The value will be the averages.
- The program also should print the data type of the averages.

Sample Input & Output

Function call	Output
extractor('ABCDEFGH12^&')	Vowels: 3 Consonants: 7 Digits: 2 Special characters: 2 Even character dictionary: {'A': 69.71....1} Odd character dictionary: {'&': 62.57....7} Even index characters average type: Float Odd index characters average type: Float

extractor('Etherea returns')	Vowels: 6 Consonants: 8 Digits: 0 Special characters: 0 Even character dictionary: {'E': 105.375} Odd character dictionary: {'n': 96.85....6} Even index characters average type: Float Odd index characters average type: Float
extractor('Abcde1')	Vowels: 2 Consonants: 3 Digits: 1 Special characters: 0 Even character dictionary: {'A': 88.33....3} Odd character dictionary: {'1': 82.33....3} Even index characters average type: Float Odd index characters average type: Float

Explanation: Let's see the third example. The string/sentence is 'Abcde1'

Here, the vowels are: A and e. So, vowel count = 2

Consonants are: b, c and d. So, consonant count = 3

Only a digit is found here. So, digit count = 1

The even indice characters are: A(0th index), c (2nd index), e (3rd index).

Hence, the ASCII sum of the alphabets = 65+99+101 = 265. Only 3 alphabets with even indices are found. As a result, the average is:  $265/3 = 88.33333333333333$  which is a float data type.

Similarly, The odd indice characters are: b (1st index), d (3rd index), 1 (5th index).

The ASCII sum of the alphabets: 98+100+49 = 247. Only 3 alphabets with odd index positions are found. So, the average is  $247/3 = 82.33333333333333$  which is a float data type.

**Objective 05:** Write a Python user-defined function named `player_status` to check the eligibility of players of a clan that is qualified on the first stage and now trying to participate in the next Call Of Duty: Mobile World Championship. A dictionary containing the UID, kills and death counts from previous qualifying matches of each player is given. Your task is to create a function that will take the player's UID as the argument and print whether the player should be on the field or bench in the World Championship. If the kill/death ratio of his

previous 3 matches is less than 2.0 then he will be on the bench. If it's 2.0 or more than 2.0 then he's eligible.

The UID, kills and deaths are given to you in the following format. The key of the dictionary is the player's name. The first element, second element and third element of dictionary values are the UID, kills in previous three matches, deaths in previous three matches respectively.

```
player = {'Stealth':[4567,(14,11,15),(8,5,7)], 'Jester':[2395,(7,6,8),(7,3,5)], 'Rajdeep':[7047,(12,12,11),(3,1,0)]}
```

### Sample Input & Output

Function call	Output
player_status(4567)	Stealth holds a K/D ratio of 2.0 Status: Field
player_status(2395)	Jester holds a K/D ratio of 1.4 Status: Bench
player_status(7047)	Rajdeep holds a K/D ratio of 8.75 Status: Field

Explanation: Calculating K/D ratio-

Add all of the kills and deaths. Divide the total kill number and death number. The resultant is the K/D ratio of the qualifying stage matches. For sample 01,

Total kills = 14+11+15 = 40

Total deaths = 8+5+7 = 20

So, K/D ratio = 40/20 = 2.0 (The player is hence eligible).

**Objective 06:** Make a program which will take as many numbers as a user wants to input, then when the user presses ENTER, a list will be created with the inputted items. If the user presses R then already given inputs will be added into a list and the program will ask the user for further inputs to make another new list. In this way, the user can make as many lists as he wants. Then, print all of the created list as another new list item. After that, make a specialized power series of the list and make a dictionary as shown in the sample input and output.

Sample Input & Output

Input	Output
1 2 3 r 7 6 r 88 99 76 ENTER	[1,2,3]  [7,6]  Final list: [[1,2,3],[7,6],[88,99,76]] Power dictionary of the final list: {1: [1, 2, 3], 2: [49, 36], 3: [681472, 970299, 438976]}

Explanation: Power series-

The final list has three elements according to the user input. These three elements have sub-elements inside them. The power series is made in a way so that it powers the sub-elements based on the index of the mother element.

The final list is: [[1,2,3],[7,6],[88,99,76]]

0th index will be ignored and the index count will start from one. When the index is 1, the sub-elements will be powered 1. When the index is 2, the next sub-elements will be squared and this series will continue until the elements come to an end.

**Objective 07:** A dictionary named elements is given to you. The key is an element name from the periodic table and the value is a tuple storing the atomic number, molar mass and density of each element of the dictionary. Write the script of a Python program which reads an element name from the user and prints the element's properties.

```
elements = {'Ac':(89,227,10.06), 'Al': (13,26.9,2.66), 'Am':(95,243,13.67)}
```

Sample Input & Output



Input	Output
Am	Am's atomic number is 95 Am's molar mass is 243 g/mol Am's density is 13.67 g/cm <sup>3</sup> (at 20 degree celsius)
Ac	Ac's atomic number is 89 Ac's molar mass is 227 g/mol Ac's density is 10.06 g/cm <sup>3</sup> (at 20 degree celsius)

**Objective 08:** A dictionary is given to you. The values are lists containing some integers. Make a new dictionary with the same keys but the values are the averages of the integers.

Sample Given & Output

Given	Output
adict= {'A':[2,3,4], 'B':[2,2,4,4], 'C':[5,6,10]}	{'A': 3.0, 'B': 3.0, 'C': 7.0}
adict={'A':[2,3,6,4], 'B':[2,9,1,4], 'C':[0,6,10]}	{'A': 3.75, 'B': 4.0, 'C': 5.333333333333333}

## Chapter 05: User Defined Function

Functions that we build ourselves to complete a specific task are known as user defined functions. Another type of Python function is built-in functions. We already learned about them. These functions exist as built-in state in the IDE and a programmer can use them whenever it's needed. We do not need to build them again. But for certain cases we might need to create a function that doesn't exist as a built-in function. In that case, we create a **user defined function** that completes the task for us. A function usually breaks down a large block of codes into a small word piece and thus we do not need to work with larger codes over and over again. This is called **functional decomposition**. Thus, program modularity becomes better if we program defining functions.

A short note: No problem analysis and exercises will be given at the end of the chapter. We already learned the basics of problem solving. By using functions, try to solve the same problems that you already solved. The aim of teaching you function is, you can make a function of each problem and later you will solve related problems just by calling the function. This saves time, trust me. A user defined function is no less than a built-in function.

Suppose, we've given a list of numbers that includes the intelligence type SI, LMI and LI of several people on a scale of 10. Our task is to average the intelligence and then get to know which person is the most intelligent among the list of people. [SI, LMI, LI respectively stands for Spatial Intelligence, Logical-mathematical Intelligence and Linguistic Intelligence.]

```
Intelligence_list = {'Naomi': [8,5,9], 'Roze': [8,8,5], 'Ray': [9,9,8], 'Stealth': [10,8,9]}
```

Without using the NumPy module, how can we solve it? Usually, we would run a for loop on the values and then count the average. After doing so we had to match the average with the keys in order to find out the person with the most intelligence. But we don't want to do the lengthy process right now. We can simply create a function that

will take lists of numbers as arguments and pass the arguments to the function's parameter and finally return us the average.

**Initial Step** : Creating a function which makes an average of numbers from a list.

```
def avg(pass_the_argument_here):
    sum = 0
    count = 0
    for i in pass_the_argument_here:
        sum += i
        count += 1
    average = sum/count
    return average
```

The code written above created a user defined function named **avg()** which returns the average of numbers from a list, string or tuples. This user defined function will work like a built-in function from now on and we can call the function whenever we need it. Pass a list to the parameters of avg() function as an argument and see the result.

```
avg([10,8,9])
9.0
```

9.0 is the output. If you're using Google Colaboratory, you can use this function even in the later code cells. Don't forget to run the original function program every time after connecting the IDE to Python 3 Google Compute Engine backend. (Precisely, you do not need to manually connect it though, it automatically connects when we run a program after logging into G-colab). Then you'll be able to use this function on your other code cells which are staying under the functional program you made earlier to create the function avg() .

OK, now let's get back into our previous problem which we kept unfinished. We created a function to calculate the average. Now, we need to separately get the average of the intelligence of each person. By running a loop over the dictionary, let's just pass the values to the avg() function's parameter and then print the keys (person names) with the

intelligence average of each person (which we'll get from **avg(values)** running inside the for loop). The full script for the problem is written below.

```
def avg(pass_the_argument_here):
    sum = 0
    count = 0
    for i in pass_the_argument_here:
        sum += i
        count += 1
    average = sum/count
    return average

Intelligence_list = {'Naomi': [8,5,9], 'Roze': [8,8,5], 'Ray':
[9,9,8], 'Stealth': [10,8,9]}

for k,v in Intelligence_list.items():
    intelligence_avg = avg(v)
    print("{}: {}".format(k,intelligence_avg))

Naomi: 7.333333333333333
Roze: 7.0
Ray: 8.666666666666666
Stealth: 9.0
```

## 5.1 Structures of a User Defined Function

A user defined function consists of a keyword (def), function's name, body, parameter list, arguments and a return statement.

```
def function_name(parameter):
    *code blocks*
    return statement
function_name(argument) [optional]
```

Figure 5.1: Basic structure of a user defined function

Every part of the function is described below.

**Keyword :** A user defined function begins with **def** keyword. A reserved keyword of Python which informs Python that a function is being started.

**Function name :** The name of the function, this is defined by the user. In our previous example, the function name was `avg()`. Function names have the same rules and conventions related to variable naming rules and conventions.

**Parameters:** Every function has parameters. There can be only one parameter or more than one. This depends on the purpose of the function and is defined by the programmer. Parameters take arguments to execute the codes written inside the body of the function.

**Arguments:** Values which are passed to the parameters are called arguments. The difference between a parameter and an argument is, the parameters are usually variables and the arguments are the values of the variables. But anyway, sometimes parameters and arguments have the same meaning.

**Function body:** Codes that will be executed to make a user defined function are written inside the function body.

**Return statement:** Return statement is necessary to finish a user defined function. Return statement displays the result when we pass an argument to the parameter. A return statement can only be executed once in a function. When a function is called, it returns a value to the point from where the function was called. By default, `None` is returned to the program if the programmer doesn't use a return statement.

```
def number(*argument):  
    return argument  
    return argument  
number(11)  
number(12)  
(12,)      #returned to the programmer
```

Figure 5.2: This figure depicts that return statement returns only one result

Return and `print()` aren't the same thing. Returned values can be stored again and reused later. But printed values cannot be reused. Return statements aren't meant to

write for human consumption but `print()` function can display the output to the human.

Return Statement	<code>print()</code> function
<pre>def number(argument):     a = 10     return a number(10) print(type(a)) &lt;class 'int'&gt; #Output</pre>	<pre>a = 10 print(a) b = print(a) print(type(b)) &lt;class 'NoneType'&gt; #Output</pre>
<pre>def number(*argument):     return argument     return argument number(11) number(12) (12,) #Output</pre>	<pre>def number(*argument):     print(argument)     print(argument) number(11) number(12) (11,) #Output (11,) (12,) (12,)</pre>

Figure 5.3: The differences between return statement and `print()` function

A returned value is reusable, as the values which are returned still hold their unique class. But printed values do not show the similar attribute. They are already displayed to humans and no longer a data type of Python. After a data type is printed, it becomes a `None` type, on whom we cannot operate anything. Furthermore, a return statement returns only one value to the console, no matter how many times a programmer uses a return statement in a function. But the `print()` function has no restriction regarding this.

## Function Call

Function call isn't a structural part of a user defined function. It has no necessity in creating a function. We call a function to use the function that we made. After building a function successfully, the function needs to be called to finish a specific task for which the function is built. Calling a function means to pass the argument(s) to the

parameter(s) of the function. We saw that a function, either built-in or user defined function, looks like “function\_name()” this. The parentheses hold the place to pass an argument. Passing an argument to the parentheses is known as function call or invocation of a function. For example: print(5), len(‘String’), sum([1,2,3]) etc.

## Types of Arguments

The arguments of a user defined function are divided into three types.

**1. Positional Arguments:** The beginning syntax of a user defined function contains a keyword, function name and the parameters of the function. These parameters are for specific arguments. While calling the function, if arguments are passed arbitrarily to the parameters then we might get invalid and undesired output. In this case, the argument order must be synchronized with the parameter orders. So, to pass an argument to a parameter if maintaining position is obligatory then the argument is known as a positional argument.

Defining function with correct order of arguments	Output
<pre>def intro(name,age,uni):      """This function prints student info"""      print("My name is",name)     print("I'm {} years old".format(age))     print("I study at {}".format(uni))  intro('Simanto',19,'BracU')</pre>	<pre>My name is Simanto I'm 19 years old I study at BracU</pre>
Defining function with wrong order of arguments	Output
<pre>def intro(name,age,uni):      """This function prints student info"""      print("My name is",name)     print("I'm {} years old".format(age))</pre>	<pre>My name is BracU I'm 19 years old I study at Simanto</pre>

<pre>print("I study at {}".format(uni))  intro('BracU',19,'Simanto')</pre>	
--	--

Figure 5.4: Usage of positional arguments

Arguments are mapped with parameters maintaining the proper order. So, invoking the function with the wrong order of parameters usually results in wrong, funny and meaningless output.

**2. Default Arguments:** It's possible to set a default argument with a parameter. The argument automatically maps with the specified parameter. For default arguments, it's needless to pass the argument to the parameter during function call. But if the programmer wishes to overwrite the default argument with his custom argument then he can specify a new argument to overwrite the default argument. The custom argument will be mapped with the ordered parameter during execution.

Defining function with a default argument	Output
<pre>def intro(name,age,uni = 'BracU'):      """This function prints student info"""      print("My name is",name)     print("I'm {} years old".format(age))     print("I study at {}".format(uni))  intro('Simanto',19)</pre>	<pre>My name is Simanto I'm 19 years old I study at BracU</pre>
Overwriting a default argument	Output
<pre>def intro(name,age,uni = 'BracU'):      """This function prints student info"""      print("My name is",name)     print("I'm {} years old".format(age))</pre>	<pre>My name is Simanto I'm 19 years old I study at BUET</pre>



```
print("I study at {}".format(uni))
intro('Simanto',19,'BUET')
```

Figure 5.5: Usage of default arguments

One more thing, default arguments should be fixed to the last parameter (every default argument should be followed by non-default arguments.) Going against the rule will raise an error.

```
def intro(name = 'Simanto',age,uni):
    """This function prints student info"""
    print("My name is",name)
    print("I'm {} years old".format(age))
    print("I study at {}".format(uni))
intro(19,'BUET')
```

[Output]

SyntaxError: non-default argument follows default argument

### Dealing with uncertain number of arguments

Imagine a function to return the product of multiple numbers. We'll never know how many numbers a user will pass to the parameter of the function. Here the number of parameters is unknown. So, it's impossible to get the correct result if the programmer limits the number of parameters. To face this situation, an asterisk sign is put before the parameter's variable. Python will understand that the current defined function isn't limited to a fixed number of parameters. Figure 5.3 shows an example regarding this. Below the Python script for the recent problem is given.

```
def pod(*parameter):
    product = 1
    for i in parameter:
        product *= i
    return product
```

```
pod(2,3,5)
```

```
Output: 30
```

```
pod(3,4,7,6)
```

```
Output: 504
```

## Chapter 06: Hand Simulation

In 2011, psychologist Howard Gardner proposed the Multiple Intelligence Theory where he mentioned about eight types of intelligence of a human. Logical-mathematical intelligence is one of them. [4] This intelligence has two portions, logical and mathematical. When we write a program to finish a specific task, we build a logic (algorithm) and proceed following the algorithm. Our logical intelligence helps us to do so. The other part, mathematical intelligence doesn't really have a great impact on this task. Hand simulation is the thing where mathematical intelligence overwhelms logical intelligence. Hand simulation (also known as **Tracing**) refers to manually executing a source code's output using pen and paper instead of using a computer. It is the reverse process of programming. In our last chapters, problems were given and we solved it by coding. We made sure that our code matches the outputs. In hand simulation, the source code will be given and we need to find the outputs. Usually, tracing is practiced in academic programming curriculum. Tracing doesn't have a good educational value on a programmer's experience but it's quite helpful for beginners to know how a program actually works. This chapter is basically an exercise chapter where certain source codes will be given and you need to find out the output of the codes. One hand simulation is done below for your understanding.

### Source Code

```
dict1 = {'a':59, 'b':-82, 'c':5, 'd':-81, 'e':53}
for i in Dict1:
    j = 0
    k = 22
    while j < 5:
        if j % 2 == 0:
            k = dict1[i]+j-(8 + k % 6)/3
            dict1[i] = dict1[i]+int(k)
        else :
            k = dict1[i]+j - (6 - k % 8) * 3
            dict1[i] = dict1[i]-int(k)
        j += 1
    print(int(k))
    print(i + "->" + str(dict1[i]))
```

To find the output, we need to clearly understand the schematic representation of the code.

## Schematic Representation

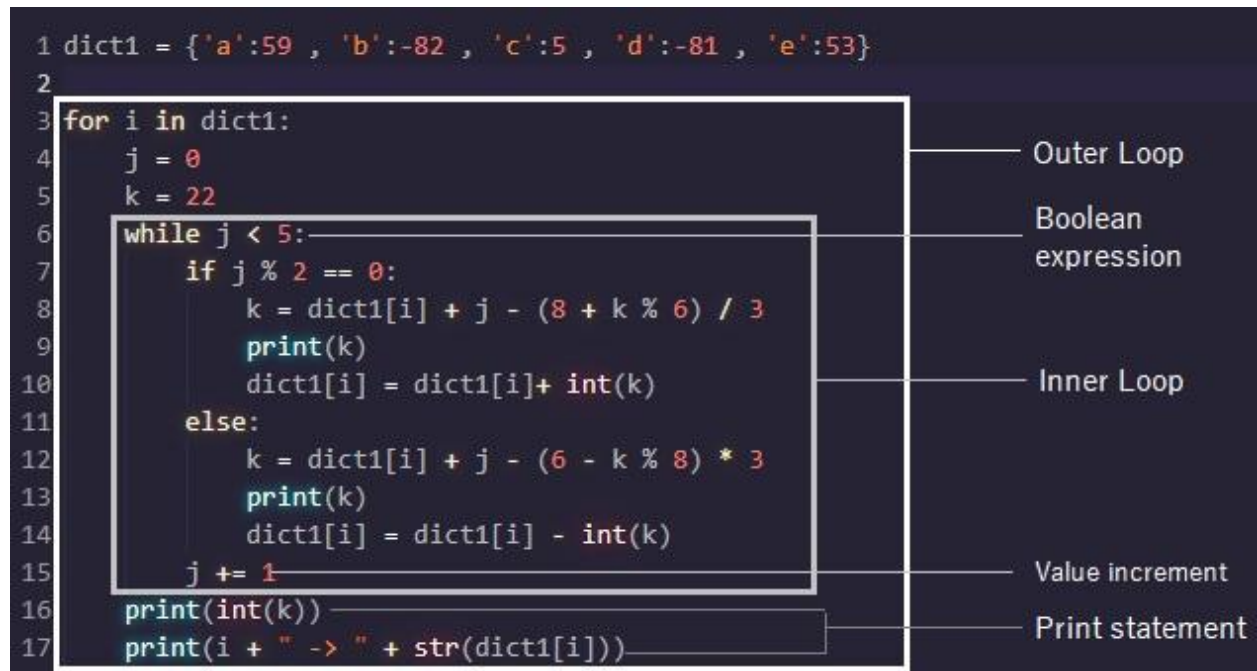


Figure 6.1: Schematic representation of the given source code

## Steps To Get The Outputs

**Step 01:** Take a clear look at the source code and find out the scopes of the **loops**. We'll be able to figure out the inner and outer loop in this way.

**Step 02:** Count the variables and make a trace table for them. This is the most important part of hand simulation. Carefully track and update the values of the variables. One simple mistake may result in wrong outputs and this will be a massive waste of time, energy and mental stability.

**Step 03:** Start executing from the first line. Python executes codes from first line to last line, left to right.

**Step 04:** If multiple loops are there, finish executing the inner loop, then move to the outer loop for execution.

## Trace Table

There are two loops, three variables and several dictionary key,value pairs are in the source code.

Variables: i, j, k

Where i stores the dictionary keys.

j stores integer values that make a boolean expression for the while loop.

K stores numeric values that can be negative or non-negative numbers (including float).

Record the initial values of every variable in the first row of the trace table. During execution, we'll keep updating the values in new columns.

i	j	k	dict1[i]	Output
'a'	0 (initial)	22 (initial) 55.0 118.0 -6.0 -19.0 8.66	dict1['a'] = 59 dict1['a'] = 114 dict1['a'] = -4 dict1['a'] = -10 dict1['a'] = 9 dict1['a'] = 17	8 a -> 17
'b'	1	-86.0 -179.0 10.0 12.0 10.33	dict1['b'] = -82 dict1['b'] = -168 dict1['b'] = 11 dict1['b'] = 21 dict1['b'] = 9 dict1['b'] = 19	10 b -> 19
'c'	2	1.0 -8.0 12.0 23.0 2.66	dict1['c'] = 5 dict1['c'] = 6 dict1['c'] = 14 dict1['c'] = 26 dict1['c'] = 3 dict1['c'] = 5	2 c -> 5
'd'	3	-85.0	dict1['d'] = -81 dict1['d'] = -166	-7 d -> -14

		-174.0 7.33 22.0 -7.0	dict1['d'] = 8 dict1['d'] = 15 dict1['d'] = 7 dict1['d'] = -14	
'e'	4	49.0 88.0 12.0 23.0 2.66	dict1['e'] = 53 dict1['e'] = 102 dict1['e'] = 14 dict1['e'] = 26 dict1['e'] = 3 dict1['e'] = 5	2 e -> -5

Trace table

So, the final output table:

Output
8
a -> 17
10
b -> 19
2
c -> 5
-7
d -> -14
2
e -> 5

## Algorithm Visualization

The tracing is explained rigorously by the following algorithm visualization. When we run the source code, Python virtual machine will start executing the code from the beginning, left to right. The time it will reach line 3, the program will step at the for loop. This loop will extract the keys of the given dictionary and store the keys in **i** variable.

For the first iteration, the value of **i** is 'a'. With this value, the program will move to line 4; then line 5. Two more variables are assigned there (**j** and **k** whose value is 0 and 22 respectively).

The machine reaches line 06 for the first time, enters into a while loop. Currently,

**j** = 0 (*First iteration of while loop*)

The boolean expression, while **j** < 5: becomes True.

**Line 07:** **j** = 0, so **j** % 2 == 0 becomes True. The machine goes under the if block for execution.

**Line 08:** **k** = dict1[**i**]+**j**-(8+**k**%6)/3

Parentheses have precedence over any other operators. Inside parentheses, modulus operator have precedence over other operators. Hence, the arithmetic operation of this line:

$$\begin{aligned} k &= 59+0 - (8+22\%6) / 3 \\ &= 59+0 - (8+4) / 3 \\ &= 59 - 12 / 3 \\ &= 59 - 4.0 \\ &= 55.0 \end{aligned}$$

This is the new value of **k** which updated the initial value. Initially, **k** = 22. Currently, **k** = 55.0.

**Line 09:** Intentionally left blank.

**Line 10:** dict1[**i**] = dict1[**i**]+int(**k**)

Here, the value of the corresponding key will be updated. We're still at our first iteration of the outer for loop. Here the value of **i** is still 'a' and dict1[**i**] refers to dict1['a']. As a result,

$$\text{dict1}['a'] = 59 + \text{int}(55.0)$$

So, dict1['a'] = 114

The key,value pair updated. The current dictionary is: {'a':114, 'b':-82, 'c':5, 'd':-81, 'e':53}

**Line 11:** Intentionally left blank.

Python won't execute Line 12 to Line 15 as it already executed the codes under the if block.

**Line 17:** The value of  $j$  incremented by +1. Current value of  $j$  is 1, which is less than 5. Therefore the while loop continues again. Python starts executing codes from line 6. (*Second iteration of while loop*)

$j = 1$ , the value is odd. The machine jumps to line 12 to execute codes of the else block.

**Line 13:**  $k = \text{dict1}[i] + j - (6 - k \% 8) * 3$

Here,  $\text{dict1}[i] = \text{dict1}['a'] = 114$

$$j = 1$$

$$6 - k \% 8 = 6 - 55.0 \% 8 = 6 - 7.0 = -1.0$$

$$\begin{aligned} \text{So, } k &= 114 + 1 - (-1.0) * 3 \\ &= 114 + 1 + 3.0 = 118.0 \end{aligned}$$

The value of  $k$  was 55.0. Now that's overwritten by the current value (118.0)

**Line 15:**  $\text{dict1}[i] = \text{dict1}[i] - \text{int}(k)$

$$\text{dict1}['a'] = \text{dict1}[i] - \text{int}(k)$$

$$= 114 - 118$$

$$= -4$$

The dictionary updated again. Current dictionary,  $\{'a':-4, 'b':-82, 'c':5, 'd':-81, 'e':53\}$

Then again the machine will reach line 17 and increment the value of  $j$  by +1.

Current value of  $j = 2$  (even value, the program will go to line 07's if block for execution).

The python virtual machine returns to line 06, then line 07 and then line 08.

**Line 08:**  $k = \text{dict1}[i] + j - (8 + k \% 6) / 3$

$$= -4 + 2 - (8 + 118.0 \% 6) / 3$$

$$= -4 + 2 - (12) / 3$$

$$= -4 + 2 - 4.0$$

$$= -6.0 \text{ (k's value updated and overwritten 118.0)}$$

**Line 10:**  $\text{dict1}[i] = \text{dict1}[i] + \text{int}(k)$

$$\text{dict1}['a'] = -4 + (-6) = -10$$

Current dictionary:  $\{'a':-10, 'b':-82, 'c':5, 'd':-81, 'e':53\}$



Machine steps line 17 again. Now,  $j = 3$ . (Odd value)

**Line 13:**  $k = \text{dict1}[i] + j - (6 - k \% 8) * 3$

Here,  $\text{dict1}[i] = \text{dict1}['a'] = -10$

$$j = 3$$

$$(6 - k \% 8) * 3 = (6 - -6.0 \% 8) * 3 = (6 - 2.0) * 3 = 4.0 * 3 = -12.0$$

$$\text{So, } k = -10 + 3 - 12.0 = -19.0$$

This is the current value of  $k$ .

**Line 15:**  $\text{dict1}[i] = \text{dict1}[i] - \text{int}(k)$

$$\text{dict1}['a'] = \text{dict1}[i] - \text{int}(k)$$

$$= -10 + 19$$

$$= 9$$

The dictionary updated again. Current dictionary,  $\{'a':9, 'b':-82, 'c':5, 'd':-81, 'e':53\}$

**Line 17:**  $j += 1$ . Now,  $j = 4$  (Even value).

The machine steps to line 06, then line 07 and executes line 08.

**Line 08:**  $k = \text{dict1}[i] + j - (8 + k \% 6) / 3$

$$= 9 + 4 - (8 + -19.0 \% 6) / 3$$

$$= 13 - (8 + 5.0) / 3$$

$$= 13 - 13.0 / 3$$

$$= 8.6666666666666668$$

The value is rounded to 8.66 ( $k$ 's value updated)

**Line 10:**  $\text{dict1}[i] = \text{dict1}[i] + \text{int}(k)$

$$\text{dict1}['a'] = 9 + (8) = 17$$

Current dictionary:  $\{'a':17, 'b':-82, 'c':5, 'd':-81, 'e':53\}$

$j$  increments again, but the first while loop ends as the boolean expression for the current value of  $j$  is False. So the machine jumps to line 18 and then line 19.

**Line 18:**  $\text{print}(\text{int}(k))$

By executing this, we get our first output: 8

**Line 19: `print(i + “->” + str(dict1[i]))`**

By executing this, we get our second output: a -> 17

We got our first two outputs. The program now will get back to the for loop. The for loop will iterate for the second time. The value of `i` will be ‘b’. When the machine heads up to line 06, the while loop will run again. Then everything will be executed following the exact manner shown above. In the whole processing, for loop will iterate only 5 times and while loop will iterate 25 times. All of the iterations will keep yielding the outputs. The for loop will end up in a total iteration of 5 times and while loop will iterate 25 times.

## Exercises

Use pen and papers to solve the following problems. You can use calculators if you want. Make trace tables or do draft works if it's necessary.

## Tracing

Some source codes are given. Find the outputs.

**1.**

```
alist = [12,5,-7,9,13,18]
blist = []
i = 0
j = 5
while i < 5:
    alist[i] = j + alist[i+1] + bool(6)
    j -= 1
    blist.append(alist[i])
    print(alist[i])
    while j > 0:

        blist[i] = alist[i] + 2
        blist[i] = blist[i] + True
```

```

        blist.append(blist[i])
        j -= 1
    i += 1

print(blist)

```

## 2.

```

dict1 = {'a': 59, 'b': 69, 'c': True}
alist = []
k = 0
j = -1
new_dict = {}
for i in dict1:
    alist += [dict1[i]]
    print(alist)
    while k < 10:

        if k % 2 == 0:
            stealth = dict1[i] + (dict1[i] - 5) - dict1[i] % -3.5
            stealth %= -2.5
            print("Stealth:",stealth)

        else :
            dict1[i] = False + dict1[i] + 5 + stealth
            new_dict.update({'Etherea':dict1[i]-stealth})
            print(1+ j<k)
            j += 1

    k += 1
print(new_dict)

```

### 3.

```
soldiers = ['Stealth','RayLe','Ivtisum','Ghost','Moonstone']
count = 0
for i in soldiers:
    for j in range(len(i)):

        if i[0] != 'S':
            sum = 0
            sum += ord(i[j])
        else:
            if count < 1:
                position = 'not a soldier'
                print(i + str(' is'),end=" ")
                print(position)
                count += 1

    if i[0] != 'S':
        if sum % 2 == 0:
            print('Soldier'+': '+ i +str('-->')+ ' Command'+': '+
Deploy'.upper()+str('-->')+ ' Code No. {}'.format(sum))
        else :
            print('Soldier'+': '+ i +str('-->')+ ' Command'+': '+
Base'+str('-->')+ ' Code No. {}'.format(sum))
```

### 4.

```
dict1 = {'a':[1,3,5], 'b':(2,6,8)}
new_dict = {}
sum = 0
i = 0
while i < 5 :
    for x in dict1:
        if i % 2 != 0:
            new_dict[x] = str(dict1[x][i-1]) + str(bool())
            print(new_dict[x])
        else:
            for j in dict1.values():
```

```

        for k in j:
            sum += k
        print(sum)
    i += 1

print(new_dict)
print(dict1)

```

## 5.

```

alist = []
i = 30
j = 3
count = 3
while i > -5:
    if i >= 15:
        j = j + i
        alist += [j]
        j += 2
    else :
        k = alist[count] % -3.3
        print(k)
        count -= 1
        store = str(k)
        num = store[3]
    i -= 5
print(alist)
print(num)

```

**6.**

```
alist = [1,4,3,7,9,5,6,7,66,45,34]
length = len(alist)
i = 0
while i < length:
    if i == 3 or i == 5:
        pass
        print("-->")
        alist[i] = alist[i] + i
        print(alist[i])

    i += 1

for i in range(len(alist)):
    if i == 4 or i == 6:
        continue
        print("-->")
    print(alist[i],end="x")
print()
print(alist)
```

## Debugging

The scripts written below have some bugs. As a result, they aren't providing the correct outputs. Try to find the errors in the script and fix the bug.

**1.** Below a script is given which was written by a programmer. This script has a bug. It is supposed to tell the maximum number and its index position in a list. But it keeps malfunctioning each time. Find and fix the error.

```
user = input("Enter numbers: ").split(",")
alist = []
for i in user:
    alist.append(int(i))
for j in range(len(alist)):
    if j==0:
        max = alist[j]
    else :
        if alist[j] < max:
            max = alist[j]
max_idx = -1
for l in alist:
    max_idx += 1
    if l == max_idx:
        break
print("Largest number in the list is {} which was found at index
{}".format(max,max_idx))
```

Sample input: 5,4,12,65,34,8

The desired output is “Largest number in the list is 65 which was found at index 3.”

But this program says, “Largest number in the list is 4 which was found at index 5.”

**2.** The program shown above was written to take 5 numbers from the user and make a list with the numbers. But it keeps showing only one list element in the output. Find the error and fix it.

```
i = 0
while i < 5:
    user = int(input("Enter numbers: "))
    alist = []
    alist += [user]
    i += 1
print(alist)
```

Sample input: 4,6,5,8,1

Desired output: [4,6,5,8,1]

Shown output: [1]





# Appendices

## Appendix 1

### Unary Operations

The term unary operation is derived from mathematics which denotes the mathematical operation of a single operand. Binary operation is the contrast operation of unary operations where two operands are operated. For instance,  $(-)$ 3 is a unary operation where 2-3 is a binary operation. Python supports three unary operators and they are +, - and ~.

**1. Unary plus:** Before a data or a variable, unary plus (+) is placed. But unary plus doesn't affect the value. The value of the variable remains the same. It works for integer, float, complex and boolean data types.

```
Var1 = 5
Var2 = +Var1
print(Var1)
```

Output

5

**2. Unary minus:** Being similar to unary plus, it's set before a variable. But it changes the data. Unary minus (-) returns the negative value of the data. It also works for integer, float, complex and boolean data types.

```
var1 = 5
Var2 = -Var1
print(Var2)
```

Output

-5

**3. Unary invert:** The symbol for unary invert is ~ which actually returns the -(variable+1) value. Unary invert is only active for integer and boolean data types.

```
Var1 = 6
Var2 = ~Var1
print(Var2)
```

Output

-7

## Appendix 2

### Built-in Functions

Below some built-in functions are discussed shortly that we might need to use.

1. count(): It returns the occurrence of an object inside a data type as an integer.

```
var1 = 'Morningstar'
print(count('r'))
```

Output

2

2. startswith(): Return True if a certain string starts with the given substring which passed as an argument of this function. Otherwise, it will return False.

```
var1 = 'Moonstone'
print(var1.startswith('M'))
```

Output

True

3. endswith(): Return True if a certain string ends with the given substring which passed as an argument of this function. Otherwise, it will return False.

```
var1 = 'Riverside'
print(var1.endswith('e'))
```

Output

True

4. `find()` : Return the index of the first appearance of the passed string argument to the function's parameter.

```
var1 = 'Ethereal'
print(var1.find('e'))
```

Output

3

5. `replace()` : This function replaces old substrings with the new string arguments.

```
var1 = 'Surreal'
var2 = var1.replace('r','l')
print(var2)
```

Output

Sulleal

The first parameter is for the string/substring that will be replaced, the second one is for the string that will replace the old substring.

6. `extend()` : It merges two lists together.

```
list1 = [1,2,4]
l2 = [5,6]
list1.extend(l2)
print(list1)
```

Output

[1,2,4,5,6]

7. `remove()` : It removes the specified list item from an existing list.

```
list1 = [1,2,3,4,5,6]
list1.remove(4)
```

```
print(list1)
```

Output

```
[1,2,3,5,6]
```

8. `index()`: This function returns the index position of a list item.

```
list1 = [1,2,3,4,5,6]
print(list1.index(3))
```

Output

```
2
```

9. `pop()`: Removes a list item based on the integer argument. The integer argument holds the index position of the item.

```
list1 = [1,2,3,4,5,6]
list1.pop(2)
print(list1)
```

Output

```
[1,2,4,5,6]
```

10. `sort()`: Returns the sorted list in ascending order. If the items aren't numeric, then it sorts based on ASCII numbers of the first characters of the items. This function works only for lists and tuples, not for dictionaries. Dictionaries have `sorted()` function that does the similar task.

```
list1 = ['Mango', 'Banana', 'pineapple', 'Apple']
list1.sort()
print(list1)
```

Output

```
['Apple', 'Banana', 'Mango', 'pineapple']
```

11. `reverse()`: It returns the list reversing every element.

```
list1 = ['Mango', 'Banana', 'pineapple', 'Apple']
list1.sort()
print(list1)
```

Output

```
['Apple', 'pineapple', 'Banana', 'Mango']
```

12. `enumerate()`: It takes a list as the argument of the function and returns a tuple including index position of each item. To display the final output `tuple()` function is used together with `print()` function.

```
list1 = ['Mango', 'Banana', 'pineapple', 'Apple']
tuple1 = enumerate(list1)
print(tuple(tuple1))
```

#### Output

```
((0, 'Mango'), (1, 'Banana'), (2, 'pineapple'), (3, 'Apple'))
```

13. `get()`: The `get()` function returns the value of a key (argument). If the key isn't present in the dictionary then it returns `None`.

```
dict1 = {'A': 32, 'B': 24}
val = dict1.get('A')
print(val)
```

#### Output

```
32
```

14. `sorted()`: This function sorts the keys or the values in ascending order and returns a new list.

```
dict1 = {'A': 32, 'B': 25, 'C': 27}
dict2 = sorted(dict1.keys())
print(dict2)
```

#### Output

```
['A', 'B', 'C']
```

Explanation: Since the keys are alphabets, they are sorted in ascending order on the basis of their position on the ASCII table.

```
dict1 = {'A': 32, 'B': 25, 'C': 27}
dict2 = sorted(dict1.values())
print(dict2)
```

#### Output

[25,27,32]

Explanation: The values are in integer type, as a result they are normally sorted in ascending order.

**del keyword:** To remove an element from a dictionary, del keyword is used.

```
dict1 = {'A': 32, 'B': 25, 'C': 27}
del dict1['C']
print(dict1)
```

Output

```
{'A':32, 'B': 25}
```

### Difference between sort() and sorted() function

sort() function	sorted() function
<p><u>Syntax:</u> list_name.sort(key, reverse = False)</p> <p>Unlike sorted(), this function can only be used on lists.</p> <p>Other attributes are similar to sorted() function.</p>	<p><u>Syntax:</u> sorted(iterable, key, reverse = False)</p> <p>Iterable = sequence or collection data type</p> <p>Key = This is optional. Used for custom sorting</p> <p>Reverse = The default value is False. For False, it sorts in ascending order. For True, it sorts in descending order.</p>

## Appendix 3

### Differences between the Data Structures

Here's a short overview on the attributes of every kind of data type and the differences between them. Having a clear knowledge about the differences will help us to choose the absolute data structure to solve a specific problem. We already are aware that all of the unique data structures

have their own idiosyncratic characteristics and it's better to find the best required one to solve a problem.

Data Types	Attributes
String	<ol style="list-style-type: none"> <li>1. Every character confined into a ' ' is a string.</li> <li>2. Immutable (can be mutated by using list).</li> <li>3. Indexing is possible. Strings are indexed by integers.</li> <li>4. Can go through a loop and are accessible. Supports membership operators (in, not in).</li> <li>5. Strings also supports overloaded operators (+, *) for concatenation.</li> </ol>
List	<ol style="list-style-type: none"> <li>1. Every character confined into a box bracket and separated by comma is a list item.</li> <li>2. Mutable.</li> <li>3. Indexed by integers. (A list item is changed/updated using indexing.)</li> <li>4. Can be run on a loop and items are accessible.</li> <li>5. Lists support every membership operators and only a overloaded operator (+)</li> </ol>
Tuple	<ol style="list-style-type: none"> <li>1. Every character confined into a first bracket and separated by comma is a tuple item. Comma separated items without the first bracket are also read as tuple items.</li> <li>2. Immutable.</li> <li>3. Indexed by integers.</li> <li>4. Can go through loops and items are accessible.</li> <li>5. Tuples support membership operators but not overloaded operators.</li> </ol>
Dictionary	<ol style="list-style-type: none"> <li>1. Key,value pairs confined into curly braces</li> </ol>



	<p>are known as a dictionary.</p> <ol style="list-style-type: none"> <li>2. Mutable.</li> <li>3. Not indexed by integers. Indexed by the keys.</li> <li>4. Can go through loops and items are accessible. The function for accessing keys and values are different.</li> <li>5. It supports membership operators but not overloaded operators.</li> </ol>
--	---

## Appendix 4

### Sorting Algorithms

Ordering an unordered data in ascending or descending order is called sorting. Python has two built-ins for sorting that we learnt earlier. Python even has several sorting algorithms to carry out the same operations that the built-in `sort()` and `sorted()` does. Two of them are discussed here.

**1. Bubble sort:** A for loop is necessary to write the bubble sort algorithm. In each iteration, adjacent elements are compared and then the elements swap their places if the first element is greater than the second element. As a result, when the loop ends all of the elements are sorted in ascending order. The algorithm is written below.

```

numbers = [7,3,1,8,9]                #Line 01
for i in range(0, len(numbers)-1):    #Line 02
    for j in range(0, len(numbers)- i - 1): #Line 03
        if (numbers[j] > numbers[j + 1]): #Line 04
            temp = numbers[j]            #Line 05
            numbers[j] = numbers[j + 1]  #Line 06
            numbers[j + 1] = temp        #Line 07

```

```

print("Sorted list:", numbers)
#Line 08
#Line 09

```

### Algorithm Visualization

One iteration is shown for your understanding.

Overview: The values of i are = 0,1,2,3,4

The values of j are = 0,1,2,3,0,1,2,0,1,0

temp = 7,7,3

#### Iteration 01:

i = 0 and j = 0

**Line 04:** j = 0, number[0] = 7, number[0+1] = 3

7 is greater than 3. Boolean expression is True.

**Line 05:** temp = 7

**Line 06:** numbers[0] = numbers[1], hence, numbers = [3,3,1,8,9] (1st time swap place)

**Line 07:** From line 06, numbers = [3,3,1,8,9]. So, numbers[j+1] = numbers[0+1] = numbers[1] = 7

So, numbers = [3,7,1,8,9]

This is how the numbers keep swapping their places. The final list will be [1,3,7,8,9]

Updated list	Iteration number
[7,3,1,8,9]	No iteration
[3,7,1,8,9]	1
[3,1,7,8,9]	2
[3,1,7,8,9]	3
[3,1,7,8,9]	4
[1,3,7,8,9]	1

### Bubble sorting

**2. Selection sort:** In selection sorting, the minimum value from the sequence is found and then it swaps the place of the first item in the sequence. Then again it finds the minimum value in the sequence from the first index and swaps it with the existing first indexed value. The list is therefore sorted when the loop ends. The algorithm of selection sort is given below.

```

numbers = [17, 3, 9, 21, 2]                                #Line 01
for index1 in range(0, len(numbers)-1):                    #Line 02
    min_val = numbers[index1]                              #Line 03
    min_index = index1                                     #Line 04
    for index2 in range(index1+1, len(numbers)):           #Line 05
        if numbers[index2] < min_val:                      #Line 06
            min_val = numbers[index2]                     #Line 07
            min_index = index2                             #Line 08
    temp = min_val                                          #Line 09
    numbers[min_index] = numbers[index1]                   #Line 10
    numbers[index1] = temp                                  #Line 11
                                                            #Line 12
print("Sorted list", numbers)                              #Line 13

```

### **Algorithm Visualization**

#### Iteration 01

#### **[Outer loop]**

index1 = 0

min\_val = 17

min\_index = 0

#### **[Inner loop]**

#### Iteration 01

index2 = 1

**Line 06:** numbers[1] = 3 which is less than the min\_val (because min\_val = 17)

**Line 07:** min\_val = 3

**Line 08:** `min_index = 1`

**Line 09:** `temp = 3`

**Line 10:** `numbers[1] = numbers[0]`

This means, `numbers = [17,17,9,21,2]`

**Line 11:** `numbers[0] = temp`

This means, `numbers = [3,17,9,21,2]` (1st swap place)

This is just the first iteration of the nested for loop. When this loop will stop iterating, we'll get the first minimum value. After that, the outer loop will give another index to run the inner loop. That loop will yield another minimum value. In this way when the inner and outer iteration will be over, the list will be sorted. The final list will be `[2,3,9,17,21]`

Updated list	Outer loop iteration
<code>[17,3,9,21,2]</code>	1
<code>[2,17,3,9,21]</code>	2
<code>[2,3,17,9,21]</code>	3
<code>[2,3,9,17,21]</code>	4

Selection sorting

## Appendix 5

### Boolean Operations

Every boolean expression returns either True or False. When this True/False builds an arithmetic equation merging with the mathematical operands, the value of True becomes 1 and the value of False becomes 0.

Boolean Expression	Output
integer + True	integer + 1 <u>Example:</u> 2 + True = 3
integer - True	integer - 1 <u>Example:</u> 2 - True = 1
integer + False	integer + 0 <u>Example:</u> 2 + False = 2
integer - False	integer - 0 <u>Example:</u> 2 - False = 2

## Solution Link

The solutions of every practice problems can be found here : [shorturl.at/zCGLV](https://shorturl.at/zCGLV) and [shorturl.at/nuOS8](https://shorturl.at/nuOS8)

## Advanced Practice Problems

### Loop, String

You can only use your learning outcomes of the concepts- branching, loop and string to solve the following problems (except the problems where secondary objectives are given. You'll have to write functions to complete the tasks fully).

1. Write a Python program to find the sum of the series  $1!/1 + 2!/11 + 3!/111 + 4!/1111 + \dots$  up to  $n$  terms. Take  $n$  as input.

Sample Input & Output

Input	Output
4	1.2124455688812124
6	1.2129495734352576

2. Write a Python program that will take a number and return whether the number is a Kaprekar number or not.

A Kaprekar number is a number whose square can be divided into two parts such that the sum of those two parts is equal to the original number and none of the parts has a value of 0.

Sample Input & Output

Input	Output
9	9 is a Kaprekar number
45	45 is a Kaprekar number
297	297 is a Kaprekar number

Explanation:  $9^2 = 81$

$$8 + 1 = 9$$

So, 9 is a Kaprekar number.

$$\text{Again, } 45^2 = 2025$$

$$20 + 25 = 45$$

So, 45 is a Kaprekar number.

$$\text{Similarly, } 297^2 = 88209$$

$$88 + 209 = 297$$

297 is a Kaprekar number.

3. Given a circle and a line segment. Find out which one of the following is true. You can use the **split()** function for this task.

- The line segment intersects the circle.
- The line segment is outside the circle.
- The line segment touches the circle.

#### Sample Input & Output

Input	Output
0 0 10 0 0 2 2	The line segment intersects the circle
4 3 10 100 100 200 200	The line segment is outside the circle.
0 0 20 -20 -100 -20 100	The line segment touches the circle.

**Instruction:** The program will give two input prompts to the user. Firstly, it will ask for the circle's center coordinates and the radius. Secondly, another input prompt will be shown to the user to input the coordinates of the two endpoints of the line segment.

- If the distance of the line segment's middle point from the circle's center is less than the radius then the line intersects the circle.
- If the distance of the line segment's middle point from the circle's center is greater than the radius then the line is outside the circle.



- If the distance of the line segment's middle point from the circle's center is equal to the radius then the line touches the circle.

Explanation: In the first sample 0 0 is the coordinate (x,y) of the circle's center and 10 is the radius. In the second line, 0 0 is the start point (a,b) of the segment and 2 2 (c,d) is the endpoint of the segment.

**Objective 04.** Write a Python program that takes a positive integer as input and removes all the digits that are greater than 9 minus the digit to their left.

Sample Input & Output

Input	Output
2572908	25205
4567	45
1234	1234

**Objective 05.** Write a Python program that takes a positive integer N as input and adds 1 to each of its even digits and removes each of its odd digits.

Sample Input & Output

Input	Output
834	95
753	0
745902	513

**Objective 06.** Look at the series below:

$$1 + \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \frac{4}{5} + \dots$$

Write a Python program that will take an integer (n) as the limit of the series and print the value of the series up to nth term.

**Constraint:** The value of n cannot be larger than 26 or less than 1 ( $1 < n < 27$ )

Sample Input & Output

Input	Output
4	The value of the series is 3.716666666666667
7	The value of the series is 6.2821428571428575
27	The valid range is $1 < n < 27$ Please try again
0	The valid range is $1 < n < 27$ Please try again

**Objective 07.** A series is given.

$$1 + 3/2 + 9/3 + 27 + 81/2 + 243/3 + 729 + \dots$$

Write a Python program to get the nth term and the value of the series. N will be inputted by the user.

Sample Input & Output

Input	Output
2	Value: 2.5 nth term: 3/2
5	Value: 73.0 nth term: 81/2
7	Value: 883.0 nth term: 729

**Objective 08.** Write a Python program that takes an integer as input and outputs the adjacent of 5 from the inputted integer.

Sample Input & Output

Input	Output
3456	6
1234	Not found
55879856	5

**Objective 09.** Write a Python program that takes an integer of more than 2 digits and powers the digit in the following manner.

Sample Input & Output

Input	Output
1234	14916
767	493649

Explanation: 1234:  $1^2 = 1$ ,  $2^2 = 4$ ,  $3^2 = 9$ ,  $4^2 = 16$ . *Final output* 14916

**Objective 10.** Write a Python program that takes an integer of more than 2 digits and powers the digit in the following manner.

Sample Input & Output

Input	Output
1234	1427256
767	736343

Explanation: 1234:  $1^1 = 1$ ,  $2^2 = 4$ ,  $3^3 = 27$ ,  $4^4 = 256$ . *Final output* 1427256

**Objective 11.** Write a Python program that will take a numeric value  $n$  as an input and print an alphabet. The numeric values start from 26 and end at 1 unlike the ASCII table. From 26 to 1 the alphabets are a to z.

**Secondary Objective:** Solve this problem by creating a function. Name the function `num_to_alph()`.

**Constraint:**  $1 \leq n \leq 26$

Sample Call & Output

Function Call	Input	Output
<code>num_to_alph(26)</code>	26	'a'
<code>num_to_alph(25)</code>	25	'b'

**Objective 12.** Write a Python program to find out the most appearing substring from several string portions. The substrings are separated by a “-” and each portion contains a substring of 8 characters. Each portion starts with a special character and appears several times in a row. You’ve to find out which portion has the most appearance of that special character. You cannot use the `split()` function.

**Constraint:** The special characters can’t appear more than 9 times in each substring.

Sample Input & Output

Input	Output
####1234-##567890-#####00	Portion 3
\$\$123456-\$\$\$\$678-\$\$\$fg567	Portion 2
@@@@DBh-@NghT-@@BDW-@@@TM	Portion 1

**Objective 13.** Write a Python program that lets a user input the products he wants and the input taking process only stops when he presses Enter without inputting the name of the product. Then count the total price for the products.

Available products: XXX = 250 Taka

YYY = 150 Taka

ZZZ = 70 Taka

Sample Input & Output

Input	Output
XXX XXX YYY	650 Taka
YYY XXX ZZZ	470 Taka

**Objective 14.** Write a Python program that takes a number as input and prints the factorial of that number in the console. Also print the sum of the numbers from 1 to the user input and print their average as well.

Sample Input & Output

Input	Output
5	Factorial: 120 Sum: 15 Average 3.0

**Objective 15.** Consider the following series:

$1 + 3 + 7 + 15 + 31 + \dots$

Your program should take an integer n and print the nth number of the pattern. Also print the summation of the series.

Sample Input & Output

Input	Output
-------	--------

3	nth number: 7 Sum: 11
6	nth number: 63 Sum: 120

**Objective 16.** Write a Python program that will take two string input and print a new string. The new string will consist of the even indexed characters of the first argument and the odd indexed characters of the second argument. In the new string, even indexed characters will be in uppercase and odd indexed characters will be in lowercase. You cannot use **upper()** and **lower()** functions for this task. But you may create some user defined functions similar to **upper()** and **lower()** functions.

**Secondary Objective:** Try solving this problem by using functions. Name the function **str\_merger()**.

Sample Call & Output

Function call	Input	Output
str_merger('abcdef','ertyu')	abcdef ertyu	'AcErY'
print(str_merger('Stealth','Et herea'))	Stealth Etherea	'SeLhTeE'

**Objective 17.** Write a Python program that will take an integer as an input. If the sum of the squared digit of the number eventually becomes 1 then the function will print 1. Otherwise it will print 0. See the samples for clarification.

**Secondary Objective:** Try solving the problem by creating a user defined function. You can name the function **squared\_one()**.

Sample Input & Output

Function call	Input	Output
squared_one(203)	203	1

squared_one(1)	1	1
squared_one(101)	101	0

Explanation:  $203 : 2^2 + 0^2 + 3^2 = 13$

$$1^2 + 3^2 = 10$$

$$1^2 + 0^2 = 1$$

Hence, The function should return 1.

On the other hand,  $101 : 1^2 + 0^2 + 1^2 = 2$

So, the function should return 0 for this number as an argument.

**Objective 18.** Write a Python program that takes an integer of more than 2 digits and powers the digit in the following manner.

Sample Input & Output

Input	Output
1234	14916
767	493649

Explanation: 1234:  $1^2 = 1$ ,  $2^2 = 4$ ,  $3^2 = 9$ ,  $4^2 = 16$ . *Final output* 14916

**Objective 19.** Create a Python program to decode some specific encrypted message. Sample messages and their meaning is given. The program will take the message as user input.

**Secondary Objective:** Make a function named **dcd()** so that a user passes the secret messages to the function's parameter and it returns the decoded message.

Sample Call & Output

Function call	Input	Output
dcd('Wt fs%nx%ymj%nrutxyjw')	Wt fs%nx%ymj%nrutxyjw	Rowan is the imposter

dcd('Jsjr~%fy%stwym3%^tz,wj%hqjfwji%mty3')	Jsjr~%fy%stwym3%^tz,wj%hqjfwji%mty3	Enemy at north. You're cleared hot.
--	-------------------------------------	-------------------------------------

**Objective 20.** Write a Python program to deconcatenate strings. The program will take two inputs from the user and should print the deconcatenated string (the uncommon strings from the both inputs). If the user inserts integers as input then the integers will be converted into strings.

**Secondary Objective:** Write a Python function to do the task. You can name the function **dcn()**.

Sample Call & Output

Function call	Input	Output
dcn(1234,12)	1234 12	'34'
dcn('7465','76')	7465 76	'45'
dcn(qwert,ew)	qwert ew	'qrt'
dcn(qsd,h)	qsd h	'None'

**Objective 21.** Write a Python program that will read dash separated integers from a user and print the characters of the integers from the ASCII table. You're not allowed to use the **split()** function to solve this problem.

Sample Input & Output

Input	Output
97-98-99-100	abcd
65-109-97-120	Amax



41-33-38-35	)!&#
-------------	------

**Objective 22.** Write a Python program which reads a string from the user and prints a new string reversing the even indexed characters. You can not use **list** data structure.

Sample Input & Output

Input	Output
Notepad	dopetaN
Whale	ehalW

Explanation: (Sample 1) The even indexed alphabets are N,t,p and d. These alphabets replaced their places with each other reversely. As a result, d and N replaced their places, p and t replaced their places.

**Objective 23.** Write a Python program which will take a sentence from the user and print the length of the last word. Don't use the **split()** function.

**Constraint:** There can be no punctuations in the sentence and only one whitespace between the words can exist.

Sample Input & Output

Input	Output
His name is Dane	Last word: Dane Length: 4
He is my best friend	Last word: friend Length: 6

**Objective 24.** Write a Python program that will take an input from the user and display whether it is a number, word or mixed with digit and letters.

If all the characters are numeric values, print DIGIT. If they are all letters, print ALPH. If it is mixed, print MIXED.

Constraint: Only numbers and alphabets can be inputted.

Sample Input & Output

Input	Output
65Yu	MIXED
0101	DIGIT
Panda	Word

**Objective 25.** A strong password must contain the following things:

- Uppercase letters (A-Z)
- Lowercase letters (a-z)
- Digits (0-9)
- Special characters (ASCII decimal range 33-47, 64, 94-95, 126)
- Length > 6

Write a Python program that will take a random password from the user and print how strong the password is. The program should also print the missing conditions.

Every criteria satisfies	Password Strength: Very Strong
One criteria misses	Password Strength: Strong
Two criteria misses	Password Strength: Good
More than two criteria misses	Password Strength: Weak
If the length is less than 6	Password Strength : Very Weak (No need to judge other criterias)

Sample Input & Output

Input	Output
aZ#9	Very Weak The password is too short

aZ#9top	Very Strong All criteria satisfied
kiroalexa	Weak Uppercase letters are missing Digits are missing Special characters are missing
kirOaleXa	Good Digits are missing Special characters are missing
kirOaleXa#	Strong Digits are missing

**Objective 26.** Write a Python program that will read four strings from the user and concatenate the common characters of the four strings. The concatenated strings will be displayed as output. The strings are mixtures of numbers and letters only. Your program should not be a case-sensitive program. If any letters are in uppercase then convert them into lowercases. You can use the **lower()** function to do so.

#### Sample Input & Output

Input	Output
Ghost-A1 HellDane9X StealthAX Etherea	ha
78Thabc 87hatCB a97O8BdT6 117a8thCCb	78tab

## List

Concepts need to be learned: Branching, Loop, String & List

**Objective 01.** A list of integers and a target is given. If two integers in the list sums up and equals the target, then print their indices.

Sample Given & Output

Given	Output
Nums = [3,2,4] Target = 6	[1,2]
Nums = [2,7,11,15] Target = 9	[0,1]

Explanation:  $2+4 = 6$ . 2 and 4 respectively staying at 1st and 2nd index (Sample Given 1)

**Objective 02.** A list will be given to you. Write the Python code of the program to get the median of the list.

**Secondary Objective:** Create a user defined function to solve this problem. Name the function **med()**.

Sample Given & Output

Function call	Given	Output
med([1,2,3,4,5])	[1,2,3,4,5]	3
med([4,6,8,0])	[4,6,8,0]	7

**Objective 03.** Write a Python program that reads a given tuple (or you can use **list** instead, it's up to you. I used tuples.) and sums and subtracts items sequentially shown in the following manner.

**Secondary Objective:** Write a user defined function to do the task. Name the function **sequence\_sumsub()**.

Sample Call & Output

Function call	Given	Output
sequence_sumsub((1,4,3,2))	(1,4,3,2)	4
sequence_sumsub((4,4,4,6,2))	(4,4,4,6,2)	8

Explanation:  $1+4-3+2 = 4$

$4+4-4+6-2 = 8$

**Objective 04.** Write a Python program where the program will take two inputs from the user. One is an integer and the other is a list of alphabets. The integer is the range of English alphabets. If any alphabet that exists between the range is missing in the inputted list then the program will print the alphabets as output. See samples for clarification. Your program should work for every sample.

Sample Input & Output

Input	Output
a,b,c,d 6	Existing alphabets: ['a','b','c','d'] Alphabets to add: ['e','f']
[b,d] 8	Existing alphabets: ['b','d'] Alphabets to add: ['a','c','e','f','g','h']
[g,h] 4	No alphabet is existing between the range Alphabets to add: ['a','b','c','d'] Alphabets to remove: ['g','h']

**Objective 05.** A list is given. Create a new list with the items that have 'a' on it.

Sample Given & Output

Given	Output
list1 = ['Apple', 'Banana', 'Text', 'Novice',	['Apple', 'Banana', 'Zodiac']

'Zodiac']	
list2 = ['1123', 'Sun', 'Moon']	[ ]

**Objective 06.** Write the Python script of a program that will take a list of items and concatenate all of the items and print the concatenated string.

**Secondary Objective:** Solve the problem using function. Name the function **str\_con()**.

Sample Call & Output

Function call	Given	Output
str_con([1,2,3,True,False,0.5])	[1,2,3,True,False,0.5]	'123TrueFalse0.5'
str_con(['8',9,int(6.5), 3>2])	['8',9,int(6.5), 3>2]	'896True'

**Objective 07.** Write a Python program where a list will be given and the program will print a new list with the summation of the numbers. (See samples). Make sure that your program works for every list. If any item of the list (passed argument) isn't numeric then the program should print None (no output will be displayed).

**Secondary Objective:** Try to solve the problem using a user defined function. You can name the function **sum\_sequence()**.

Sample Given & Output

Function call	Given	Output
sum_sequence([5,3,4,2,8,7])	[5,3,4,2,8,7]	[8,12,14,22,29]
sum_sequence([5,8,'a'])	[5,8,'a']	

**Objective 08.** You're given a n x n matrix. Rotate the matrix as shown in the samples.

**Secondary Objective:** Make a user defined function to solve this problem. You can name this function **rotate\_matrix()**

Sample Call & Output

Function call	Given	Output
<code>rotate_matrix([[1,2,3], [4,5,6], [7,8,9]])</code>	<code>[[1,2,3], [4,5,6], [7,8,9]]</code>	<code>[[7,4,1],[8,5,2],[9,6,3]]</code>
<code>rotate_matrix([[4,5,6,7],[3,3, 4,5],[4,5,5,5],[2,3,4,1]])</code>	<code>[[4,5,6,7],[3,3,4,5],[4,5,5,5],[ 2,3,4,1]]</code>	<code>[[2,4,3,4],[3,5,3,5],[4,5,4,6],[ 1,5,5,7]]</code>

**Objective 09.** Write a Python program where you're given a list of words. The user will input a word with some mistakes and the program will print the word that has the most alphabetical matches with the given word. You can use the `lower()` function if you want.

Sample Given & Output

Given & Input	Output
<pre>given = ['Evening', 'Morning', 'Afternoon',         'Night'] Input: avaning</pre>	Evening
<pre>given = ['Evening', 'Morning', 'Afternoon',         'Night'] Input: Orni</pre>	Morning
<pre>given = ['Evening', 'Morning', 'Afternoon',         'Night'] Input: igH</pre>	Night

**Objective 10.** You'll be given a list of pin numbers of 4 digits. Write a Python program that will take a 2 digit integer number and match it with the pin numbers. If every digit of the integer belongs to a certain pin of the list then the program will create a new list for the pin numbers. If the integer contains the same digits then the program will create the new list judging the first digit. Finally, print the list.

```
given = [2032, 4354, 6821, 4567, 9086, 1023, 1011]
```

Sample Input & Output

Input	Output
23	[2032,1023]
11	[6281,1023,1011]
97	[ ]

**Objective 11.** Write a Python program that will read some words (the first alphabet of each word will be in uppercase) and other alphabets will be in lowercase without any space and make a list with the words.

Sample Input & Output

Input	Output
StealthEthereaJamesDane	['Stealth', 'Etherea', 'James', 'Dane']
RiverCloudSky	['River', 'Cloud', 'Sky']

**Objective 12.** Write a Python program that will take a list from the user and print the subtracted value of the numeric items (integer and float type). Note that the list is given. The user need not to input the list to the program.

Sample Input & Output

Given	Output
['Lily', '23', 23, 'Rose', '1.85']	23
['a', True, 6<7, '5', 7, '8', 9, 2.5]	-4.5

**Objective 13.** You're given two lists. Make two more new lists with squaring the numbers from the given lists. Then, sum the even indexed numbers of the first new list and subtract the odd indexed numbers of the second new list. You'll get two resultants from the new lists. Then sum the two resultants. Finally, print the sum.



Sample Given & Output

Given	Output
List1 = [2,4,6,8] List2 = [1,9,5,7]	72
List1 = [10,11,14,15,20] List2 = [21,15,18,22]	437
List1 = [1,1,1,1] List2 = [1,1,1,2]	-1

**Objective 14.** Write a Python program that will take a string (single length) and print the previous and next five characters of the string from the ASCII table.

Sample Input & Output

Input	Output
'f'	Previous 5: ['e', 'd', 'c', 'b', 'a'] Next 5: ['g', 'h', 'i', 'j', 'k']
'y'	Previous 5: ['x', 'w', 'v', 'u', 't'] Next 5: ['z', '{', ' ', '}', '~']
'K'	Previous 5: ['J', 'I', 'H', 'G', 'F'] Next 5: ['L', 'M', 'N', 'O', 'P']

**Objective 15.** Write a Python program that can do the following task shown in the explanation.

Given	Output
list1 = [10,20,15,40] list2 = [12,22,42,32]	193
list1 = [11,12,21,14,15,16]	124

list2 = [2,6,5,3,9,10]	
list1 = [45,62,78,96,55,99] list2 = [61,43,23,24]	643

Explanation:

**Given 01:**

**list1 operation**

$$10 + 40 = 50$$

$$20 + 15 = 35$$

$$\text{list1\_new} = [50, 35]$$

**list2 operation**

$$12 + 32 = 44$$

$$22 + 42 = 64$$

$$\text{list2\_new} = [44, 64]$$

$$\text{final\_sum} = 50 + 35 + 44 + 64 = 193$$

Given 02:

**list1 operation**

$$11 + 16 = 27$$

$$12 + 15 = 27$$

$$21 + 14 = 35$$

$$\text{list1\_new} = [27, 27, 35]$$

**list2 operation**

$$2 + 10 = 12$$

$$6 + 9 = 15$$

$$5 + 3 = 8$$

$$\text{list2\_new} = [12, 15, 8]$$

$$\text{final\_sum} = 27 + 27 + 35 + 12 + 15 + 8 = 124$$

**Constraint:** The given lists' length cannot be an odd integer and the items can be only integers.

**Objective 16.** A list will be given to you which contains some string elements. Some of the strings are concatenated with each other. They are concatenated by a single dash. Write a Python program that will decompose the concatenated strings and print a new list with them. You can use the built-in **split()** function for this objective.

Sample Input & Output

Input	Output
['CSE110', 'CSE111-CSE230', 'ENG101', 'ENG102-PHY111']	['CSE110', 'CSE111', 'CSE230', 'ENG101', 'ENG102', 'PHY111']
['Cat-Dog', 'Apple-Banana', 'Pen', 'Paper', 'PDF']	['Cat', 'Dog', 'Apple', 'Banana', 'Pen', 'Paper', 'PDF']

**Objective 17.** Write a Python program where a list will be given to you. The list contains the mixture of string, integers, float numbers and boolean data type. Your program will print a new list converting every existing element into integers. If the given list's certain item isn't integer already, then the program should convert it into a string first. After that, it will get the string's ASCII summation and place it in the new list.

**Secondary Objective:** Create a Python function to solve the task. You can name the function **str\_to\_int()**.

Sample Call & Output

Function call	Given	Output
str_to_int([100, 20, 'abc', '30', True, 2>3,])	[100, 20, 'abc', '30', True, 2>3,]	[100, 20, 294, 99, 416, 491]
str_to_int(['***', 19.5, 5==7, 20, True + 1, False+3])	['***', 19.5, 5==7, 20, True + 1, False+3]	[126, 205, 491, 20, 2, 3]

**Note:** Expressions are solved first if they exist inside the data structures. 2>3, True + 1, False + 1 won't show similar behavior like integers or strings. Rather, they'll be treated like False (boolean), 2 (integer) and 3(integer) respectively. One more thing, when True and False are placed inside an arithmetic expression, the values for True becomes 1 and False becomes 0.

**Objective 18.** Write a Python program that gives N times input prompt to the user and then displays a list with the inputted numbers' subtraction. N will be given by the user as well. See samples for clarification. You're allowed to use the **split()** function.

Input	Output
4 6 7 8 10 9 5 7 9 9 9 13 21 32	-9 -11 -9 -40 [-9, -11, -9, -40]
3 1 2 3 4 5 4 3 2 1 3	-13 -1 -2 [-13, -1, -2]

**Objective 19.** Write a python program that takes a number sequence as input and prints whether it is a UB Jumper or Not UB Jumper. Input receiving process will stop after getting "BREAK" as input.

Concept: Let there are N numbers in a list and that list is said to be a UB Jumper if the absolute values of the difference between the successive elements take on all the values 1 through N – 1. For example, 2 1 4 6 10 is a UB Jumper because the absolute differences between them are 1 3 2 4 which is all numbers from 1 to (5 - 1) or 4.

You cannot use **abs()** function to complete this objective. You are allowed to use the **split()** function.

Sample Input & Output

Input	Output
1 4 2 3 2 1 4 6 10	UB Jumper UB Jumper

1 4 2 -1 6 BREAK	Not UB Jumper
---------------------	---------------

# Dictionary

Concepts need to be learned: Branching, Loop, String, List, Dictionary

**Objective 01.** A dictionary will be given to you. The keys of the dictionary are tuples containing integers and the values are alphabetical strings. Your program should print a new dictionary where the values will be the average of the integers (integer converted) inside the tuple and the keys will be the last character of the string.

**Secondary Objective:** Create a user defined function named **str\_avg\_dct()** to solve this problem. The function should **return** the new dictionary.

Sample Call & Output

Function call	Given	Output
str_avg_dct({(2,5,4): 'asdf', (1,0,1): 'Taskforce', (9,6,0,5): 'Eight'})	{(2,5,4): 'asdf', (1,0,1): 'Taskforce', (9,6,0,5): 'Eight'}	New dictionary: {'f': 3, 'e': 0, 't': 5}

**Objective 02.** Write a Python function that will take (given) a dictionary from the user. Keys of the dictionary are strings and the values are tuples and lists of numbers (integer and floats). Your program should print a new dictionary where there will be only a key and a value. The key is a string holding the ASCII summation of the given dictionary's key strings and the value is the average of the given dictionary's values.

**Secondary Objective:** Write a Python function to solve the problem. Name the function **single\_dict()**.

Sample Call & Output

Function call	Given	Output
single_dict({'String':(2,5,4),	{'String':(2,5,4),	{1851: 7.444}

'Integer':[10,10], 'Float': (6,9,8,13))}	'Integer':[10,10], 'Float': (6,9,8,13))}	
---	---	--

**Objective 03.** Nn. You'll be given a dictionary. Write the Python program to print a single sub-dictionary of the given dictionary that contains the longest (largest length) value. You can use the built-in **max()** function for this task if you want.

Sample Given & Output

Given	Output
{'a': ('*', '-'), 'x': ['a', 'q', 'u', 'A'], 'Num': (2,3,4)}	{'x': ['a', 'q', 'u', 'A']}
{1: [10,20,30], 3: [20,30,40,50], 2: (1,)	{3: [20,30,40,50]}

**Objective 04.** You're given a list made of integer and string data types. Write a Python script which will print a dictionary where the integer will be the values and the leftover strings will be the keys. If the appearance count of numeric and string items do not match then it would be impossible to make the dictionary. There will be no output.

**Secondary Objective:** Create a user defined function to solve the problem. Name the function **list\_to\_dict()**.

Sample Call & Output

Function call	Given	Output
list_to_dict([100, 200, 'a', 250, 'b', 'c'])	[100, 200, 'a', 250, 'b', 'c']	{'a': 100, 'b': 200, 'c': 250}
list_to_dict(['100', 100, 200, 'd', 'x', 800, 700])	['100', 100, 200, 'd', 'x', 800, 700]	
list_to_dict(['100', 100, 200, 'd', 'x', 800])	['100', 100, 200, 'd', 'x', 800]	{'100': 100, 'd': 200, 'x': 800}

**Objective 05.** In the previous problem, you made a dictionary. Now write another program that will do the same thing but the values will be in reverse order.

**Secondary Objective:** Try solving it by creating a user defined function. You can name this function `reverse_list_to_dict()`

#### Sample Call & Output

Function call	Given	Output
<code>list_to_dict([100, 200, 'a', 250, 'b', 'c'])</code>	<code>[100, 200, 'a', 250, 'b', 'c']</code>	<code>{'a': 250, 'b': 200, 'c': 100}</code>
<code>list_to_dict(['100', 100, 200, 'd', 'x', 800, 700])</code>	<code>['100', 100, 200, 'd', 'x', 800, 700]</code>	Invalid list. The numbers of strings and integers do not match.
<code>list_to_dict(['100', 100, 200, 'd', 'x', 800])</code>	<code>['100', 100, 200, 'd', 'x', 800]</code>	<code>{'100': 800, 'd': 200, 'x': 100}</code>

**Objective 06.** Write a Python program that will read (given) a dictionary and print a new dictionary where the keys will be turned as values and the values will be the keys.

#### Sample Given & Output

Given	Output
<code>{'a': 35, 'b': 36, 'f': 30}</code>	<code>{35: 'a', 36: 'b', 30: 'f'}</code>

**Objective 07.** You're given a dictionary. Print the key with the biggest length.

#### Sample Given & Output

Given	Output
<code>{'Serena': 18, 'Eth': 19, 'Stealth': 21, 'Dane': 25, 'Ghost': '35+'}</code>	Stealth

{‘DS E-Tense’: 1, ‘Chevrolet’: 2, ‘Saleen S1’: 3, ‘Porsche’: 5, ‘BMW’: 6}	DS E-Tense
---	------------

**Objective 08.** A dictionary will be given to you which consists of single lengthed strings as keys and values are tuples with integers. But be aware that the tuples might have some strings inside them too.

Write a Python program that will create a new dictionary where the keys will stay the same (There’s a miscellaneous case by the way) . In the new dictionary, the values of the corresponding keys will be either the average or the summation of integers inside each tuple. If the tuple length is an even integer then the value of the key will be the summation of the integers. If the tuple length is an odd integer then the value of the key will be the average of the integers. Moreover, if any string exists in any of the tuples, the string will be concatenated with the key. Your program should work for every given list.

#### Sample Given & Output

Given	Output
{‘a’: (5,10,5), ‘b’: (11,12,13,14), ‘c’: (9,8,3,’a’), ‘d’: (3,4,3,9,5,’x’,’y’)}	{‘a’: 6.66, ‘b’: 50, ‘c-a’: 20, ‘d-yd-x’: 4.8}
{‘a’: (5,10,5), ‘b’: (11,12,13,14), ‘c’: (9,8,3,’a’), ‘d’: (3,4,3,9,5,’x’,’y’,’z’)}	{‘a’: 6.66, ‘b’: 50, ‘c-a’: 20, ‘d-zd-yd-x’: 24}
{‘a’: (5,10,5), ‘b’: (11,12,13,14,’0’,’8’,’a’), ‘c’: (9,8,3,’a’), ‘d’: (3,4,3,9,5,’x’,’y’,’z’)}	{‘a’: 6.66, ‘b-ab-8b-0’: 12.5, ‘c-a’: 20, ‘d-zd-yd-x’: 24}

Note: You may ignore the output sample for averages. You don’t need to round the average value. Keep it as it is.

**Objective 09.** Write a Python program that will take a word from the user and print the ASCII summation of the lowercase characters and the ASCII multiplication of uppercase characters. These data will be printed via a dictionary. Keys will be the first uppercase and lowercase letter of the input respectively. See samples for details.

#### Sample Input & Output



Input	Output
DraGOn	{‘D’: 381412, ‘r’: 321}
StArs	{‘t’: 172640, ‘S’: 345}
lost	{‘l’: 0, ‘o’: 450}
PEN	{‘E’: 430560, ‘P’: 0}

**Objective 10.** Write a Python program that will keep printing numbers from 1 to n. The program will take input from the user. No matter what the user inputs, for his/her every input your program should print numbers consequently. When the user will press Enter, the program will be stopped and display the dictionary of the user’s entered inputs and the outputs.

Sample Input & Output

Input	Output
A	1
b	2
fgfg	3
etwdgd	4
lll	5
	{‘A’:1, ‘b’: 2, ‘fgfg’: 3, ‘etwdgd’: 4, ‘lll’:5}

**Objective 11.** You’re given a list of words. Some of the words are mixed with uppercase and lowercase letters. Write a Python program that will return a dictionary consisting of the corrected words as keys and list of uppercase and lowercase letters as values.

Sample Given & Output

Given	Output
[‘apple’, ‘baNANA’, ‘pineapple,	{‘banana’: [‘ba’, ‘NANA’], ‘watermelon’:

<code>['wAtErMelON', 'MANgo']</code>	<code>['wtrel', 'AEMON'], 'mango': ['go', 'MAN']}]}</code>
<code>['fRUits', 'auDaCITy', 'glasses', 'temPo']</code>	<code>{'fruits': ['ruits', 'RU'], 'audacity': ['auay', 'DCIT', 'tempo': ['temo', 'P']</code>

**Objective 12.** Write a Python program that will read (given) a list from the user and print a dictionary. See the sample input and output below.

Sample Given & Output

Given	Output
<code>list1 = ['Alex:', 400, 'Dane:', 150, 'Rin:', 200]</code>	<code>{'Alex': 400, 'Dane': 150, 'Rin': 200}</code>
<code>alist = ['Jones:', 100, 'Jack:', 25, 'Riyan:', 45, 'Lee:', 250]</code>	<code>{'Jones': 100, 'Jack': 25, 'Riyan': 45, 'Lee': 250}</code>

**Objective 13.** Write a Python program that will read (input) a list from the user and print a dictionary. See the sample input and output below.

Sample Given & Output

Input	Output
<code>['Alex': 400, 'Dane': 150, 'Rin': 200]</code>	<code>{'Alex': 400, 'Dane': 150, 'Rin': 200}</code>
<code>['Jones': 100, 'Jack': 25, 'Riyan': 45, 'Lee': 250]</code>	<code>{'Jones': 100, 'Jack': 25, 'Riyan': 45, 'Lee': 250}</code>

**Objective 14.** Look at the series below:

$$1 + \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \frac{4}{5} + \dots$$

Write a Python program that will take an integer (n) as the limit of the series and print the value of the series up to nth term. The program also will print a dictionary where the keys will be lowercase letters (starts from a, ends in z) and the values will be terms of the series.

**Constraint:** The value of n cannot be larger than 26 or less than 1 ( $1 < n < 27$ )

Sample Input & Output

Input	Output
4	The value of the series is 3.716666666666667 Dictionary: {'a': '1/2', 'b': '2/3', 'c': '3/4', 'd': '4/5'}
7	The value of the series is 6.2821428571428575 Dictionary: {'a': '1/2', 'b': '2/3', 'c': '3/4', 'd': '4/5', 'e': '5/6', 'f': '6/7', 'g': '7/8'}
27	The valid range is $1 < n < 27$ Please try again
0	The valid range is $1 < n < 27$ Please try again

## User Defined Functions

Concepts need to be learned: Branching, Loop, String, List, Tuple, Dictionary

**Objective 01.** Python can sum (concatenate) strings and list but they cannot subtract those. Create a Python function with two parameters that will take two list arguments and if any common element exists between the lists, it will make a new list removing the common elements and return the new list. You can name the function **uncommon()**.

Sample Call & Output

Function call	Output
<code>print(uncommon([1,2,3,4],[2,3]))</code>	<code>[1,4]</code>
<code>print(uncommon(['q', 's', 'd'], ['q', 's', 'd']))</code>	<code>[]</code>
<code>uncommon([5,3,4,8,9],[a,b,3,4])</code>	<code>[5,8,9,a,b]</code>

**Objective 02.** Write a Python function named **overloaded()** that will have three parameters. It will take two string arguments and return the multiplied/concatenated number/string. The remaining parameter is a default parameter whose argument is 1.

- If both strings are numeric (integer only) then the program should return the multiple of the numbers. If the default argument is changed then the resultant will be concatenated to the resultant default argument times.
- If both strings are non-numeric then the program should return the concatenated string default argument times.
- If one string is numeric and the other one is non-numeric then the program should return the concatenated string as well. Default argument's functionality stays the same as mentioned before.

#### Sample Call & Output

Function call	Output
overloaded(22,3)	'66'
overloaded('abc','def')	'abcdef'
overloaded('a','bc',3)	'abcabcabc'
overloaded('ar',23)	'ar23'
overloaded('22','3',2)	'6666'

**Objective 03.** Strings are immutable in Python. Create a function so that we can make the strings mutable. The function will receive three arguments in the parameters. The first argument will be the string and the second argument will be the index position. The last argument is the character string that we want to replace with the existing character on the index position. You can name this function **str\_mutator()**

#### Sample Call & Output

Function call	Output
str_mutator('Etherea', 6, 'al')	'Ethereal'

str_mutator('Heaven', 5, 'nly')	'Heavenly'
str_mutator('Red Dragon', 3, 'x')	'RedxDragon'
str_mutator('River', 1, 1)	'R1ver'

**Objective 04.** Tuples are immutable in Python. Create a function so that we can make the tuples mutable. The function will receive three arguments in the parameters. The first argument will be the tuple and the second argument will be the index position. The last argument is the item that we want to replace with the existing character on the index position. You can name this function **tuple\_mutator()**

#### Sample Call & Output

Function call	Output
tuple_mutator((2,3,5,8),2, 'a')	(2,3,'a',8)
tuple_mutator(('Peace', 'Moon', 'Celestial'),0, 'Echo')	('Echo', 'Moon', 'Celestial')

**Objective 05.** Python cannot return the keys+value length of a dictionary. Create a user defined function called **hybrid\_len()** that can return how many items are there in a dictionary.

#### Sample Call & Output

Function call	Output
hybrid_len({'a': (1,2,['x','y','z'],4), 'b': [9,8,('y','u','i','o'),65,4]})	11
hybrid_len({'apple': 4, 'banana': 5, 'mango': 6})	6

**Objective 06.** Python has no built-in function to get a character's position on a negative indexing system. Write a Python function named **neg\_idx()** that will return the negative index position of a string. There will be two arguments. The first one is for the whole string and the

second one is for the character. If any integer/float or other data type is passed to the parameter then they will be converted into strings first.

#### Sample Call & Output

Function call	Output
<code>neg_idx('Mr. String', 'S')</code>	-6
<code>neg_idx('A rabbit', '')</code>	-7
<code>neg_idx(1234, 3)</code>	-2

**Objective 07.** Dictionary items are unordered. Which means, they aren't accessible by indexing. Dictionaries have no attribute called indexing. Since, we can't do indexing on dictionary items, create a user defined function named **dict\_idx()** that will take a dictionary and the index number in a parameter. After that, the function should return a single lengthened dictionary (only one key,value pair) that was existing on the index position.

**Rules:** While making the function, don't forget to count the index from 0. The index will increment by +1 after passing each key,value pair. (Just like the indexes of sequence data types)

#### Sample Call & Output

Function call	Output
<code>dict_idx({'a': ('*', '-'), 'x': ['a', 'q', 'u', 'A'], 'Num': (2,3,4)},2)</code>	<code>{'Num': (2,3,4)}</code>
<code>dict_idx({1: [10,20,30], 3: [20,30,40,50], 2: (1,)},0)</code>	<code>{1: [10,20,30]}</code>

**Objective 08.** Create a function named **dict\_to\_list()** that will receive a dictionary argument and returns a list with the keys and values of the dictionary. In the list, keys and values should be placed adjacently.

#### Sample Call & Output

Function call	Output
<code>dict_to_list({'Pen': 2, 'Paper': 3, 'Notepad': 1})</code>	<code>[Pen, 2, Paper, 3, Notepad, 1]</code>
<code>dict_to_list({'Stx': (10,20), 'Eth': (11,), 'D9X': (13,12,16)})</code>	<code>['Stx', (10,20), 'Eth', (11,), 'D9X', (13,12,16)]</code>

**Objective 09.** Create a Python function named **str\_checker()**. The function will receive a list/tuple as an argument and return True if the sequence has any string data type. Else, it will return False.

Sample Call & Output

Function call	Output
<code>str_checker((1,2,3,'t', 3&gt;2))</code>	<code>True</code>
<code>str_checker([True, 2.5, 100])</code>	<code>False</code>

**Objective 10.** Create a Python function named **middle()** that will return the middle item of a list or tuple. If the length of the list is an even integer, then the middle item can't be found. So the function will return the concatenated strings of pre-middle and post-middle items.

Sample Call & Output

Function call	Output
<code>middle([1,2,3,4])</code>	<code>'2-3'</code>
<code>middle(('Abs', 'len', 'append', 'max', 'min'))</code>	<code>'append'</code>
<code>middle(('Newton', 'Einstein', 'Rutherford', 'Bohr'))</code>	<code>'Einstein-Rutherford'</code>

**Objective 11.** Write a Python function named **spc\_remover()** that will take a sentence as an argument. There will be arbitrary spaces in the sentence. Your program should replace the multiple spaces with a single space.

### Sample Call & Output

Function call	Output
<code>spc_remover('His name is Bill')</code>	<code>'His name is Bill'</code>
<code>spc_remover('No mercy')</code>	<code>'No mercy'</code>

**Objective 12.** Write a Python function named **dct\_str()** that will take a dictionary as an argument and return a new dictionary where the keys will be the string concatenation of every key and the value will be the string concatenation of the values. If any numeric string (ASCII range 48-57) or digit exists in the given dictionary's key/value then the numbers will be summed up instead of string concatenation. The summed up numbers will be later concatenated with the string type items. Make sure your program works for every given dictionary.

**Constraint:** No float numbers can be given as key or value to the given dictionary.

### Sample Given & Output

Given	Output
<code>dct_str({'a': 59, 'b': 69, 'c': 'Yes', 'd': True, 'e': '49', 1: 'No', 2: False})</code>	<code>{'abcd3': 'YesTrueNoFalse177'}</code>
<code>dct_str({'R': 1, 2: '3', 'F': a})</code>	<code>{RF2 : a4}</code>

**Unique Objective 01:** Is your father/mother a service holder? Does he/she need to do rigorous calculations and he/she feels so devastated? If your answer is Yes, then ask him/her what they actually do. Take notes, then make a user defined function for him/her and send the Google colab file to him/her so that his/her struggle comes to an ease. Make it too easy so that he/she doesn't find it too hard to use your program. If your answer is No, then ask somebody else who does the rigorous calculations in their job.



**Unique Objective 02:** Make a calculator that will count your running CGPA throughout the whole semester. Your faculty must be provided the evaluation process and which evaluation carries how much marks. Then make a calculator where you can input your quiz, assignment, attendance or other marks and the program will display your current CGPA.

How to calculate the running CGPA?

Imagine, you are a student of the first semester and you finished 3 quizzes, 4 assignments with full attendance. Sum up the marks and convert the mark into CGPA according to your grading policy.

**Unique Objective 03:** Write a Python program that can solve two quadratic equations.

(a)  $4x + 3y - 2 = 0$ ;  $x + 2y - 3 = 0$

(b)  $2x + 3y = 4$ ;  $x - y = 7$

(c)  $5x + 2y - 11 = 0$ ;  $3x + 4y - 1 = 0$

Answers

(a)  $x = -1, y = 2$    (b)  $x = 5, y = -2$    (c)  $x = 3, y = -2$

## References

- [1] Nørmark, Kurt (2011). Overview of the Four Main Programming Paradigms.  
[https://homes.cs.aau.dk/~normark/prog3-03/html/notes/paradigms\\_themes-paradigm-overview-section.html](https://homes.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigm-overview-section.html)
- [2] Python Software Foundation (n.d). General Python Faq. Retrieved April 8, 2023, from [shorturl.at/gyCZ1](https://shorturl.at/gyCZ1)
- [3] Kernighan, B. W (1972). A Tutorial Introduction to the Language B. Bell Laboratories.  
<https://www.bell-labs.com/usr/dmr/www/btut.pdf>
- [4] Cherry, K (2023). Gardner's Theory of Multiple Intelligences. Verywell mind.  
<https://www.verywellmind.com/gardners-theory-of-multiple-intelligences-2795161>