Input, Processing and Output and displaying output with Print function

* The input source is the keyboard, and the output destination is the terminal display. The python shell itself is such a program; its inputs are python expressions or statements.
* The programmer can also force the output of a value by using the "print" function.

Syntax:-

```
print (<expression>)
```

Example:-

```
>>> print ('Hello World')
```

Output:-

```
Hello World
```

In this above example, the text 'Hello World' is the text that we want python to display.

You can also write a "print" function that includes two or more expressions seperated by commas. In such case, the "print" function evaluates the expressions and display their results, seperated by single quotes, in one line.

Syntax:-

```
print (<expression>, ..., <expression>)
```

If you create Programme in python, you'll often want your Programme to ask the user for input. You can do this by using the "input" function. This function causes the program to stop and wait for the user to enter a value from the keyboard.

The following Example receives an input string for the user and saves it for further Processing.

Example:-

```
>>> name = input("Enter your name!")
Enter your name: Sowmya Datta
>>> name
'Sowmya Datta'
>>> Print(name)
Sowmya Datta
```

The input function does the following:

1. Displays a prompt for the input. In this Example, the Prompt is
"Enter your name:"
2. Receives a string of keystrokes, called characters entered at the keyboard and returns the string to the Shell.

G. Sowmya

The string returned by the function in our example is saved by assigning it to the variable name. The form of an assignment statement with the 'input' function is the following:

< variable identifier > = input (< a string prompt >)

A variable identifier, or variable for short, is just a name for a value. When a variable receives its value in an input statement, the variable then refers to this value. If the user enters the name "Soumya Datta" in our last example, the value of the variable 'name' can be viewed as follows:

```
>>> name
'Soumya Datta'
```

* The 'input' function always builds a string from the user's keystrokes and returns it to the program. After inputting strings ~~to that appropriate numeric~~ representent numbers, the programmer must convert them from strings to the appropriate numeric types.

```
>>> input
>>> a = int (input ("Enter the first number"))
>>> b = int (input ("Enter the second number"))
```

>>> Print ("the sum is", a + b)

Output:-

Enter the first number : 10

Enter the second number : 5

The sum is 15

Comments:-

* comments are the non-executable statements in a Program. They are just added to describe the statements in the Program code.
* comments make the Program Easily readable and understandable by the Programmers as well as others Users who are seeing the code.
* The interpreter simply ignores the comments.

Example:- Program to use comments

>>> #This is a comment

>>> Print ("Hello") # to display Hello

>>> # Program Ends here

Output:

Hello

* Note that the three Comments in the Program are not displayed. You can also type a comment in a new

line (or) on the same line after a statement (or) expression.

Note:- A Program can have any number of comments.

* comments can be placed at the end of a line, and python will ignore the rest of the line:

Example:-

```
print("Hello, World!") #This is a comment
```

Output:-

```
Hello World
```

* A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

Example:-

```
#print("Hello World!")
print("Sowmya Datta")
```

output:-

```
Sowmya Datta.
```

Multiline Comments :-

* python does not really have a syntax for multi_line comments.

* To add a multiline comment you could insert a '#' for each line:

Example:-

```
#This is a comment
#written in
#more than just one line
Print("Hello, World!")
```

Output:-

```
Hello, World!
```

* you can add a multiline string (triple quotes) in your code, and place your comment inside it.

Example:-

```
"""
This is a comment
written in
more than just one line
"""
Print("Hello, World!")
```

Output:-

```
Hello, World!
```

———— * ———— G. Soumya

Variables :-

* Variables are containers for storing data values.

Creating Variables :-

* python has no command for declaring a variable.
* A variable is created the moment you first assign a value to it.

Example :-

```
x = 5
y = "John"
Print (x)
Print (y)
```

Output :-

```
5
John
```

* Variables do not need to be declared with any particular type and can even change type after they have been set.

Example :-

```
x = 4       # x is of type int
x = "Sowmya"  # x is now of type string
print (x)
```

Output:-

Soumya

Get the type of Variable:-

You can get the data type of a variable with the type() function.

Example:-

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Output:-

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

* String variables can be declared either by using single (or) double quotes:

Example:-

```
x = "John"
y = 'John'
```

Output:-

```
John
John
```

G. Soumya

Case-Sensitive:-

* Variable names are Case-sensitive

Example:-

```
a = 4
A = "Sowmya Datta"
```

A will not overwrite a

Variable names:-

A variable can have a short name (like x and y) or more descriptive names (age, carname, total_volume).

Rules for python variables:

→ A variable name must start with a letter or the underscore character

→ A variable name cannot start with a number

→ A variable name can only contain alphanumeric characters and underscores (A-Z, 0-9, and _)

→ Variable names are case-sensitive (age, Age and AGE are three different variables)

Example: Legal variable names:

```
myVar = "Sowmya Datta"
my_Var = "Sowmya Datta"
_my_Var = "Sowmya Datta"
myVar = "Sowmya Datta"
```

MYVAR = "Sowmya Datta"

myVar2 = "Sowmya Datta"

Example:- Illegal Variable names

2myVar = "Sowmya Datta"

my-Var = "Sowmya Datta"

my Var = "Sowmya Datta"

Many Values to Multiple Variable:-

python allows you to assign values to multiple Variables in one line:

Example:-

```
x, y, z = "Orange", "Banana", "cherry"
print(x)
Print(y)
Print(z)
```

Output:-

Orange
Banana
cherry

One Value to Multiple Variable:-

you can assign the same value to multiple Variables in one line:

```
>>> x = y = z = "Orange"
Print(x)
Print(y)
```

Output:-

orange
Orange
Orange

G.Sowmya

Reading Input from the Keyboard:-

* input():- this function first takes the input from the user and then evaluates the expression, which means python automatically identifies whether user entered a string or a number or list. If the input provided is not correct then either syntax error or exception is raised by python.

Example:-

```
Val = input("Enter your value :")
Print (Val)
```

Output:-

```
Enter your value : 123
123
```

How the input function works in python:-

* When input() function executes program flow will be stopped until the user has given an input.
* The text or message display on the output screen to ask a user to enter input value is optional.

Example:-

```
num = input ("Enter number :")
Print (num)
```

```
name1 = input("Enter name:")
print(name1)
print("type of number", type(num))
print("type of name", type(name1))
```

Output:-

```
Enter number: 123
123
Enter name: Soumya Datta
Soumya Datta
type of number <class 'int'>
type of name <class 'str'>
```

*

Performing Calculations:-

Example:-

```
val1 = 2
val2 = 3
a = val1 + val2
print(a)
b = val1 - val2
print(b)
c = val1 * val2
print(c)
d = val1 / val2
print(d)
```

Operators in python :-

* Operators are used to Perform Operations on Variables and Values.
* python divides the Operators in the following groups:
  - → Arithmetic operators
  - → Assignment operators
  - → Comparison operators
  - → Logical operators
  - → Identify operators
  - → Bitwise operators
* Arithmetic Operators :- used with numeric values to perform common mathematical operators :

Operator	Name	Example
+	Addition	x+y
-	Subtraction	x-y
*	Multiplication	x*y
/	Division	x/y
%/%	Modulus	x%y

Python Assignment operators:-

* Assignment operators are used to assign values to variables:

Operator	Example	Same as
=	x=5	x=5
+=	x+=3	x=x+3
-=	x-=3	x=x-3
*=	x*=3	x=x*3
/=	x/=3	x=x/3
%=	x%=3	x=x%3
&=	x&=3	x=x&3
|=	x|=3	x=x|3
^=	x^=3	x=x^3

python comparison operators:-

Comparison operators are used to compare two values:

G. Soumya

Operators	Name	Example
==	Equal	x == y
!=	Not Equal	x != y
>	Greater than	x > y
<	less than	x < y
>=	Greater than or Equal to	x >= y
<=	less than or Equal	x <= y

python logical operators :- Logical operators are used to combine conditional statements.

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not (x < 5 and x < 10)

G. Soumya

Python Identity Operators :- Used to compare the objects, not if they are Equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Bitwise operators :- Bitwise operators are used to compare (binary) numbers :

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	zero fill left shift	Shift left by pushing zeros in from the right & let the leftmost bits fall off
>>	signed right shift	Shift right by pushing copies of the leftmost bit in from the left & let the rightmost bits fall off

G. Soumya

Type conversions in python :-

* the process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. python has two types of type conversion:

1) Implicit type conversion
2) Explicit type conversion

→ Implicit Type conversion :-

In implicit type conversion, python automatically converts one data type to another data type. this process doesn't need any user involvement.

Example :- converting integer to float

```
a = 123
b = 1.23
sum = a + b
print("datatype of a :", type(a))
print("datatype of b", type(b))
print("value of sum", c)
print("value of sum:", type(c))
```

G. Soumya

Output:-

```
datatype of a : <class 'int'>
datatype of b : <class 'float'>
Value of sum : 124.23
datatype of num_new : <class 'float'>
```

In the above Program,

→ We add two variables 'a' and 'b', storing the value in 'sum'.

→ We will look at the data type of all three objects respectively.

→ In the output, we can see the data type of 'a' is integer while the data type of 'b' is float.

→ Also we can see the 'sum' has a float datatype because python always converts smaller data types to larger data types to avoid the loss of data.

Explicit Type Conversion:-

In Explicit Type conversion, users convert the data type of an object to required data type. We use the Predefined functions like int(), float(), str() Etc, to perform Explicit type conversion.

G. Soumya

This type of conversion is also called typecasting because the user casts (changes) the datatype of the objects.

Syntax:-

<required_datatype> (expression)

* Typecasting can be done by assigning the required data type function to the expression.

Example:- Addition of string and integer using explicit Conversion.

```
a = 123
b = "456"
Print ("data type of a", type(a))
Print ("data type of b before Type casting:", type(b))
b = int(b)
Print ("datatype of b after typecasting", type(b))
c = a+b
Print ("Sum of a and b:", c)
Print ("Data type of c:", type(c))
```

Output:-

```
Data type of a: <class 'int'>
Data type of b before type casting: <class 'str'>
```

Data type of b after type casting : <class 'int'>

Sum of a and b : 579

Data type of the c : <class 'int'>

Expressions :-

* Expressions Provide an easy way to Peaform operations on data values to Produce other data values.

i) Arithmetic Expressions :-

An Arithmetic expression consists of operands and operators. ~~Combined in a manner that is~~

Operator	Meaning	Syntax
-	Negation	-a
**	Exponentiation	a ** b
*	Multiplication	a * b
/	Division	a / b
//	quotient	a // b
%	Remainder or Modulus	a % b
+	Addition	a + b
-	Subtraction	a - b

G. Soumya

The Precedence rules applied during the Evaluation of arithmetic Expressions in python:

→ Exponentiation has the highest precedence and is Evaluated first.

→ Unary negation is Evaluated next, before multiplication, division and remainder.

→ Multiplication, both types of division, and remainder are Evaluated before addition and subtraction.

→ Addition and Subtraction are Evaluated before assignment.

→ With two exceptions, Operations of Equal Precedence are left associative, so they are Evaluated from left to right. Exponentiation and assignment operations are right associative, so consecutive instances of these are Evaluated from right to left.

* You can use Parentheses to change the order of Evaluation

Expression	Evaluation	Value
5+3*2	5+6	11
(5+3)*2	8*2	16
6%2	0	0
2*3**2	2*9	18
-3**2	-(3**2)	-9
(3)**2	9	9

2**3 **2	2 ** 9	512
(2**3) ** 2	8**2	64
45/0	Error: Cannot divide by 0	
45 % 0	Error: Cannot divide by 0	

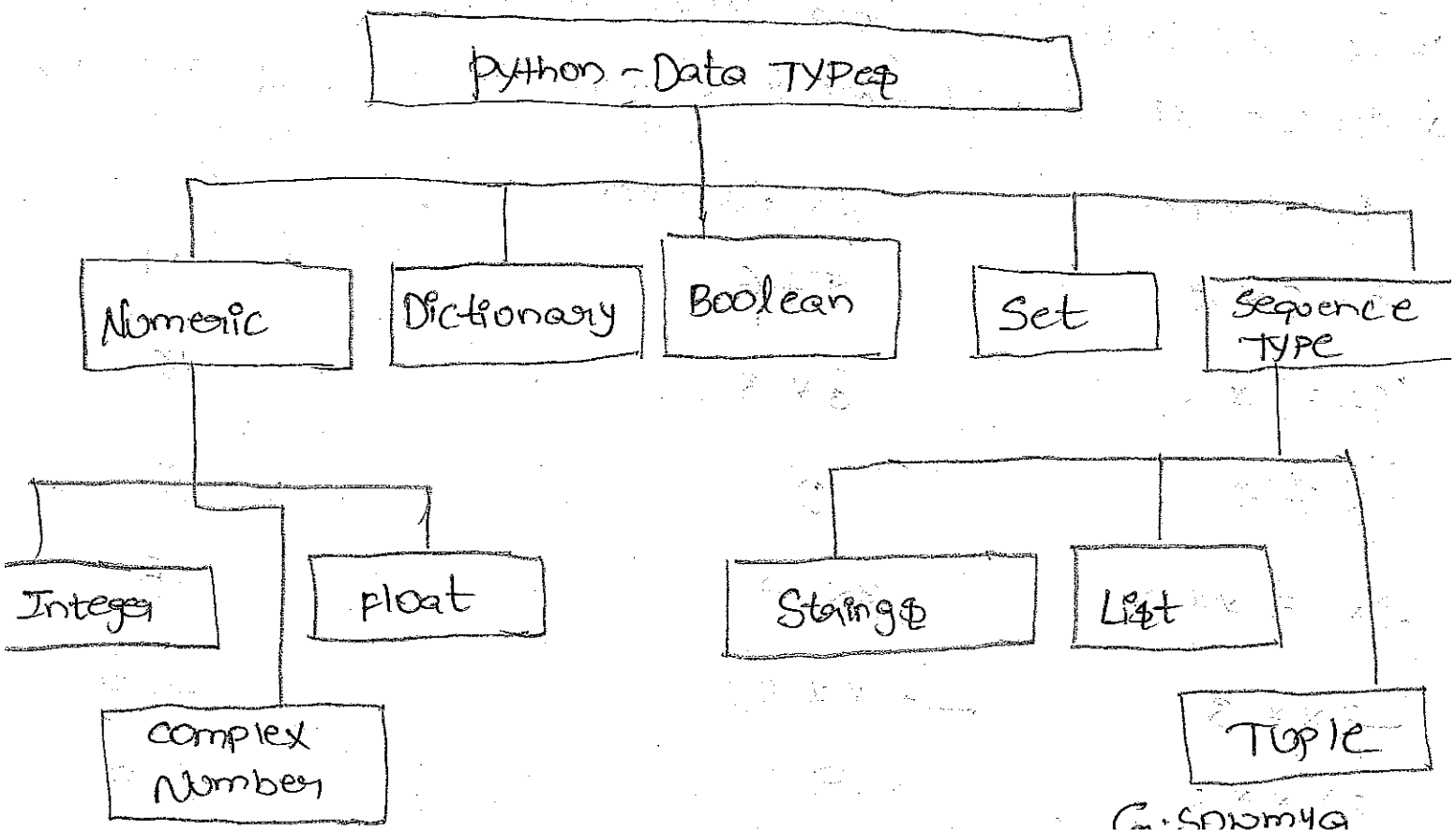
Some arithmetic Expressions and their values

Chapter - 2

DataTypes, Expressions

Python Data Types:-

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be Performed on a Particular data.



C. Soumya

Numeric:-

* In python, numeric data type represent the data which has numeric value.
* Numeric value can be integer, floating number or even complex numbers. These values are ~~ider~~ defined as int, float and complex class in python.

→ **Integers**:- This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In python there is no limit to how long an integer value can be.

→ **Float**:- This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point.

→ **Complex Numbers**:- Complex number is represented by complex class. It is specified as (real part) + (imaginary part)j. For example -2+3j

Example:-

```
a = 5
Print("Type of a:", type(a))
b = 5.0
Print("Type of b:", type(b))
```

```
c= 2+4j
Print("\nType of c:", type(c))
```

Output:-

```
Type of a: <class 'int'>
Type of b: <class 'float'>
Type of c: <class 'complex'>
```

Sequence Type:-

In python, sequence is the ordered collection of similar or different data types. Sequence allows to store multiple values in an organized and efficient fashion. There are several sequence types in python:

i) String:- A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one.

Creating String:- Strings in python can be created using single quotes or double quotes or even triple quotes.

Example:-

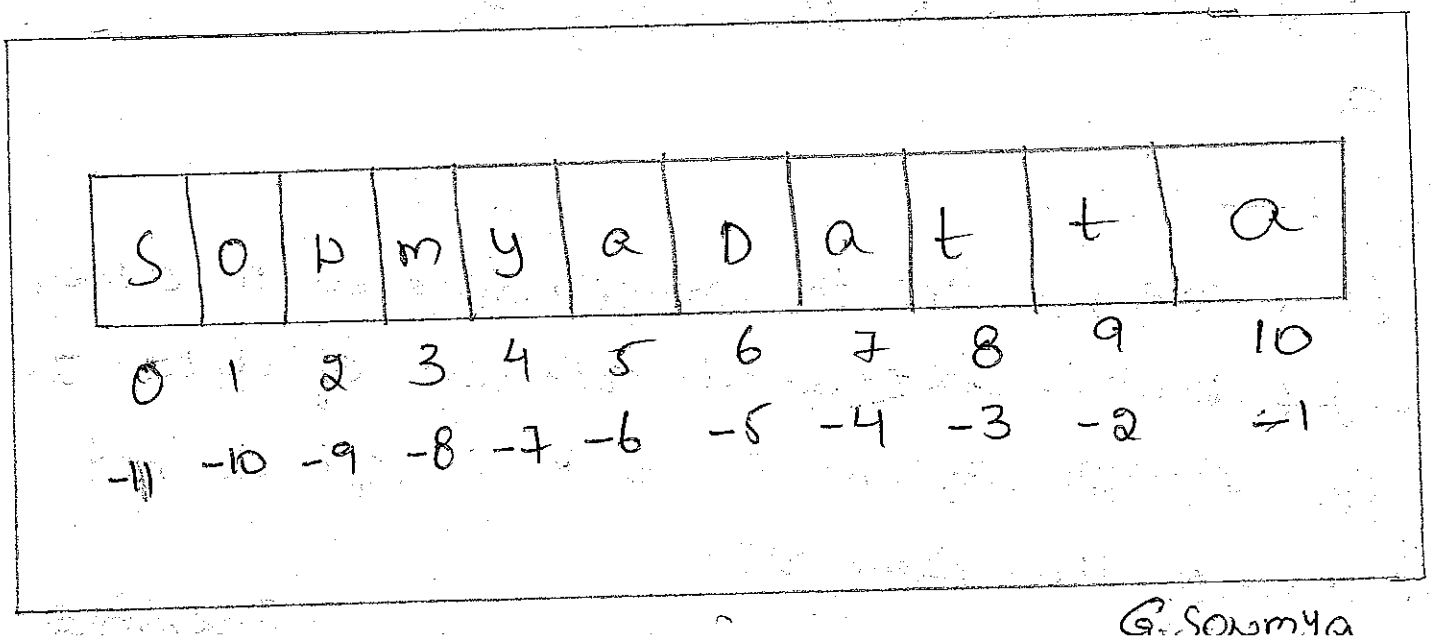
```
a = 'Soumya Datta'
Print("String with the use of Single quotes:")
Print(a)
```

```
b = " Gandla Soumya"
Print ("In string with the use of double quotes :")
Print (b)
Print (type (b))
c = """ Gandla Soumya Datta"""
Print ("In string with the use of Taiple quotes :")
Print (c)
Print (type (c))
```

Accessing Elements of string :-

In python, individual characters of a string can be accessed by using the method of indexing. Indexing allows negative address references to access characters from the back of the string

E.g:- -1 refers to the last character, -2 refers to the second last character and so on.



Example:-

```
a = "SowmyaDatta"
Print("Initial String:")
Print(a)
# Printing first character
Print("\n First character of string is:")
Print(a[0])
# Printing Last character
Print("\n Last character of string is:")
Print(a[-1])
```

Output:-

```
Initial String:
SowmyaDatta
First character of string is:
S
Last character of string is:
a
```

List :—

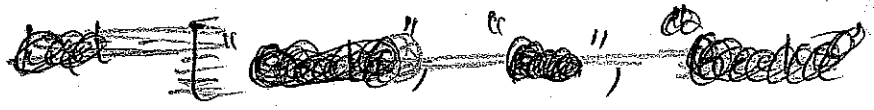
Lists are just like the arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

G. Sowmya

Creating List:-

Lists in python can be created by just placing the sequence inside the square brackets[].

```
List = []
Print ("Initial blank list:")
Print (List)
List = ['GandlaSowmyaDatta']
Print ("\n List with the use of string:")
Print(List)
```



```
List= ['Gandla', 'Sowmya', 'Datta']
Print( List[0])
Print (List[1])
List = [['Gandla', 'Sowmya'], ['Datta'])
Print ("\n Multi-Dimensional List:")
Print (List)
```

Output:-

```
Initial blank List:
[]
List with the use of string:
['GandlaSowmyaDatta']
```

List containing multiple values:

Gandla

Datta

Multi-Dimensional List:

[['Gandla', 'Soumya'], ['Datta']]

Accessing Elements of List:-

In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list. In python, negative sequence indexes represent positions from the end of the array.

Example:-

```
List = ["CSE", "IT", "DS"]
Print("Accessing element from the list")
Print(List[0])
Print(List[2])
Print("Accessing Element using negative indexing")
Print(List[-1])
Print(List[-3])
```

Output:

```
Accessing Element from the list
CSE
DS
```

Accessing Element Using Negative Indexing

CSE

DS

Tuple:-

Tuple is also an ordered collection of python objects. The only difference between tuple and list is that tuples are immutable i.e tuples cannot be modified after it is created. It is represented by tuple class.

Creating Tuple:-

Tuples are created by placing a sequence of values seperated by 'comma' with or without the use of parenthesis for grouping of the data sequence. Tuples can contain any number of elements and of any datatype.

Example:-

```
Tuple1 = ()
Print ("Initial Empty Tuple:")
Print (Tuple1)
Tuple1 = ('Yellow', 'Green')
Print ("\n Tuple with the use of String:")
Print (Tuple1)
List1 = [1, 2, 4, 5, 6].
Print ("\n Tuple using List:")
```

```
Print(tuple(list1))
Tuple1 = tuple('yellow')
Print("\nTuple with the use of function:")
Print(Tuple1)
Tuple1 = (0, 1, 2, 3)
Tuple2 = ('python', 'program')
Tuple3 = (Tuple1, Tuple2)
Print("\nTuple with nested Tuples:")
Print(Tuple3)
```

Output:-

```
Initial Empty Tuple:
()
Tuple with the use of string:
('yellow', 'Green')
Tuple using list:
(1,2,4,5,6)
Tuple with the use of function:
('y', 'e', 'l', 'l', 'o', 'w')
Tuple with nested tuples:
((0,1,2,3), ('python', 'program')
```

G. Soumya

Accessing Elements of Tuple:-

In order to access the tuple items refer to the index number. Use the index operator [] to access an item in a tuple. The index must be an integer.

Examples:-

```
tuple1 = tuple([1,2,3,4,5])
Print ("First Element of tuple")
Print (tuple1[0])
Print("\n last Element of tuple")
Print (tuple1[-1])
Print("\nThird last Element of tuple")
Print (tuple1[-3])
```

Output:-

```
First Element of tuple
1
Last Element of tuple
5
Third last element of tuple.
3
```

Boolean:-

Datatype with one of the two built in values, True or False.

Note:- True and False with capital 'T' and 'F' are valid booleans otherwise python will throw an error.

Example:-

```
Print (type (True))
Print (type (False))
Print (type (True))
```

Output:-

```
<class 'bool'>
<class 'bool'>
```

Set :-

In python, Set is an unordered collection of datatype that is iterable, mutable and has no duplicate Elements.

Creating Sets:-

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces,

separated by 'comma'.

Example:-

```
set1 = set()
Print("Initial blank set:")
Print(set1)
set1 = set("pythonProgramming")
Print("\nset with the use of String:")
Print(set1)
set1 = set(["Yellow", "Green", "Red"])
Print("\nset with the use of list:")
Print(set1)
set1 = set([1, 2, 'python', 4, 'Programming', 6, 'python'])
Print("\nset with the use of Mixed Values")
Print(set1)
```

Output:-

```
Initial blank set:
set()
set with the use of String:
{'p', 'y', 'o', 't', 'n', 'h', 'a', 'g', 'r', 'm', 'i'}
set with the use of list:
{'Yellow', 'Green', 'Red'}
```

Set with the used of Mixed Values:

{1, 2, 4, 6, 'python', 'programming'}

Accessing Elements of Sets:-

Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a loop.

Example1:-

```
set1 = set(["python", "Programming", "python"])
Print("\n Initial set")
Print(set1)
Print("\n Elements of set:")
for i in set1:
    Print(i, end=" ")
Print("python" in set1)
```

Output:-

```
Initial set:
{'python', 'Programming'}
Elements of set:
python Programming
True
```

Dictionary:-

Dictionary in python is an unordered collection of data values, used to store data values like a map, which unlike other data types that hold only single value as an element, Dictionary holds key : value pairs.

Creating Dictionary:-

In python, a Dictionary can be created by placing a sequence of elements within curly {} braces, seperated by 'comma'. Values in a dictionary can be of any datatype and can be duplicated, where as key's can't be repeated and must be immutable.

Note: Dictionary keys are case sensitive, same name but different cases of key will be treated distinctly

Example:

```
a = { 1: 'Soumya', 2: 'Datta', 3: 'soumya' }
Print ("In Dictionary with the use of Integer keys :")
Print (a)
D = { 'Name': 'Soumya', 1: [1, 2, 3, 4] }
```

Print("\n Dictionary with the use of Mixed keys:")
Print(a)

Output:-

Dictionary with the use of Integer keys:
{1: 'Soumya', 2: 'Datta', 3: 'Soumya'}

Dictionary with the use of Mixed keys:
{1: [1,2,3,4], 'Name': 'Soumya'}

Accessing Elements of Dictionary:

In order to access the items of dictionary refer to its key name. key can be used inside Square brackets.

Example:-

Dict = {1: 'Soumya', 'name': Datta', 3: 'Soumya'}
Print(Dict ['name'])

Output:

Datta

———— * ————

G. Soumya

Strings Assignment :-

* Strings are amongst the most popular types in python. We can create them simply by enclosing characters in quotes. Python treats single quotes / double quotes the same as triple quotes.
* Creating strings is as simple as assigning a value to a variable.

Example:-

```
a = 'Hello World'
b = "python Programming"
c = """NRIIT College"""
```

——— * ———

Character Sets :-

* Some Programming languages use different data types for strings and individual characters. In python, character literals look just like string literals and are of the string type. The term ASCII stands for American Standard Code for Information Technology).

The below fig. shows the mapping of character values to the first 128 ASCII codes. The digits in the left column represent the leftmost digits of an ASCII code.

and the digits in the top row are the rightmost digits. Thus, the ASCII code of the character 'R' at row 8, column 2 is 82.

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

The ASCII character set maps to a set of integers, Python's Ord and chr functions convert characters to their numeric ASCII codes and back again respectively.

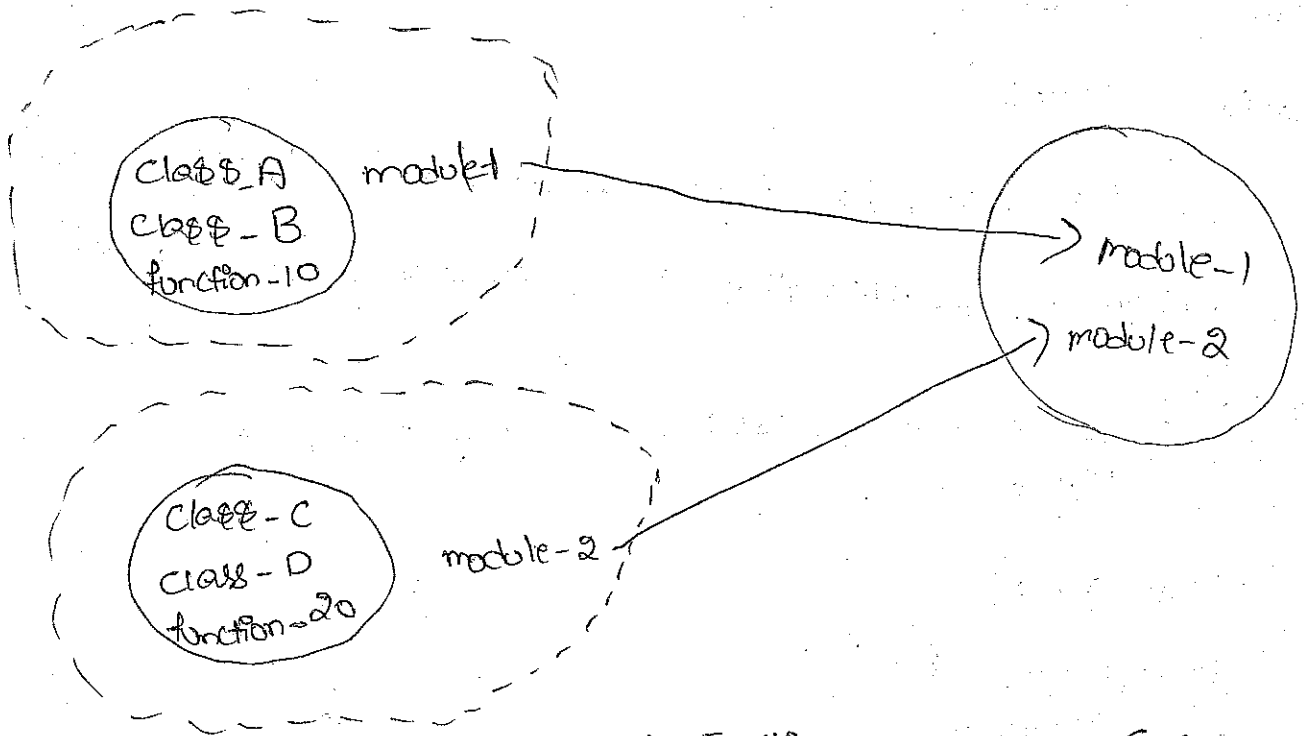
G Saanya

```
>>> ord('a')
97
>>> ord('A')
65
>>> chr(65)
'A'
>>> chr(66)
'B'
```

Using Functions and Modules:-

By seeing the below picture we can understand that package contains a collection of modules and a module contains a collection of functions.

Thus, we can say functions are the subset of modules and modules are the subset of packages.



Modules:-

A module is simply a python file with a .py extension that can be imported inside another python Program.

The module contains -

→ definitions and implementation of classes

→ Variables

→ functions that can be used inside another Program

Creating a module:-

In the below Program, a function is created with the name "module" and saving this file with name Saamya.py i.e name of the file and with Extension.py

Example:- Creating Module containing single function

```
def Module():
    Print("Hey, I am a Module")
# defining a Variable
location = "python"
```

In the below Program, we have created 4 functions for adding, multiplying, subtracting and division. Saving file as Operations.Py

Example:- Creating Module containing many functions

```
def add(x, y):
    return(x+y)
def Subtract(x, y):
    return(x-y)
```

```
def mul (x,y):
    return (x * y)
def div (x,y):
    return (x / y)
```

Importing a Module:-

Importing the function using import statement (When interpreter encounters an import statement, it imports the module if the module is present in the search path).

Example:-

```
import Sowmya
Sowmya.Module()
print(Sowmya.location) #print the variable declared
```

Output:-

```
Hey, I am a module
python
```

Example:- #importing module operations.py

```
import Operations
print(operations.add (10, 2))
print(operations.substract (15,8))
print(Operations.mul (45, 10))
print(Operations.div (90,5))
```

Output:-

```
12
7
450
18.0
```

Functions:-

* A function is a block of code which only runs when it is called.
* You can pass data, known as Parameters, into a function.
* A function can return data as a result.

Creating a function:-

In python a function is defined using the def keyword:

Example:-

```
def a():
    Print("Hello from a function")
```

Calling a function:-

To call a function, use the function name followed by parenthesis:

Example:-

```
def a():
    Print("Hello from a function")

a()
```

Arguments:-

* Information can be passed into functions as arguments.
* Arguments are specified after the function name, inside the parenthesis. You can add as many arguments as you want, just seperate them with a comma.

G. Soumya

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example:-

```
def a(fname):
    print(fname + "programming")
a("python")
a("c")
a("Java")
```

Output:-

```
python Programming
c Programming
Java Programming
```

Chapter - 3

Decision Structures and Boolean Logic:

Python if statement:-

An "if statement" is written by using the if keyword.

Example: if statement

```
a=33
b=200
if b>a:
    print("b is greater than a")
```

Indentation :-

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code.

Example:-

```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

↓ indentation

if .. Elif :- The elif keyword is python's way of saying "if the previous conditions were not true, then try this condition".

Example:-

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

if ... Else :- The else keyword catches anything which isn't caught by the preceding conditions.

Example:-

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
```

```
elif a==b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

Example:-

```
a=20
b=33
if b>a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

Nested Decision Structures:-

The nesting of decision structures allows a program to sequentially determine the current state of a problem component under investigation. Often a decision structure needs to be used to select to which category a data item belongs.

A decision structure is a construct in a computer program that allows the program to make a decision and change it's behaviour based on that decision.

A nested if is an if statement that is the target to another if statement. Nested if statements means an if statement inside another if statement.

G. Sowmya

Nested if:- You can have if statements inside if statements, this is called nested if statements.

Example:-

```
x = 41
if x > 10:
    Print ("above ten")
if x > 20:
    Print ("and also above 20!")
Else:
    Print ("but not above 20")
```

Output:-

Above ten
and also above 20!

———— * ————

Comparing Strings:-

Method 1:- Using Relational operators:

The relational operators compare the Unicode values of the characters of the strings from the zeroth index till the end of the string. It then returns a boolean value according to the operator used.

Example:-

"Soumya" == "SOUmya"
Will return True as the Unicode of all characters are Equal.

G. Soumya

Program:-

```
Print ("Sowmya" == "Sowmya")
Print ("Apple" < "apple")
Print ("Book" > "book")
Print ("@owmya" != "@owmya")
```

Output:-

```
True
True
False
False
```

Method2:- Using is and is not :-

'is' operator checks whether both the operands refer to the same object or not. The same is the case for != and is not.

Program:-

```
a = "sowmya"
b = "sowmya"
c = a
Print (a is a)
Print ( a is b)
Print ( a is c)
```

Output:-

```
True
True
True
```

Boolean Variables:-

Booleans represent one of two values: True or False.

Boolean Values:

When you compare two values, the expression is evaluated and python returns the Boolean answer:

Example:-

```
Print(10 > 9)
Print(10 == 9)
Print(10 < 9)
```

Output:-

```
True
False
False
```

Evaluate Values and Variables:-

The bool() allows you to evaluate any value, and give you True or False in return,

Example:-

```
Print(bool("Hello"))
Print(bool(15))
```

Output:-

```
True
True
```

Example:-

```
x = "Hello"
y = 15
Print(bool(x))
Print(bool(y))
```

Output:-

```
True
True
```

G. Sowmya

Chapter 4:-

Repetition Structures:-

While Loop:-

With the python while loop we can execute a set of statements as long as a condition is true.

Example:- Print i as long as i is less than 6:

```
i = 1
while i<6:
    Print(i)
    i+ = 1
```

Output:-
1
2
3
4
5

* The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

The break statement:-

With the break statement we can stop the loop even if the while condition is true:

Example:-

```
i=1
While i<6:
    Print(i)
    if i==3:
        break
    i+ = 1
```

output:-
1
2
3

The continue Statement:-

With the continue statement we can stop the current iteration, and continue with the next:

Example:-

```
i=0
while i<6:
    i+=1
    if i==3:
        continue
    Print(i)
```

Output:-

1
2
4
5
6

Example:- Print a message once the condition is false:

```
i=1
while i<6:
    Print(i)
    i+=1
Else:
    Print("i is no longer less than 6")
```

Output:-

1
2
3
4
5
i is no longer less than 6

Python for Loops:-

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

With the for loop we can execute a set of statements, once for each item in a ~~object~~ list, tuple, set etc.

G. Soumya

Example:- Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

Output:-

```
apple
banana
cherry
```

Looping Through a String:-

Even strings are iterable objects, they contain a sequence of characters:

Example:- Loop through the letters in the word "banana"

```
for x in "banana":
print (x)
```

Output:-

```
b
a
n
a
n
a
```

* With the break statement we can stop the loop before it has looped through all the items:

Example:- Exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
    break
```

Output:-

```
apple
banana
```

The range() function :-

TO loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example :-

```
for x in range(6):
    print(x)
```

output :-

```
0
1
2
3
4
5
```

* The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter; range(2,6), which means values from 2 to 6 (but not including 6):

Example :- using the start parameter

```
for x in range(2,6):
    print(x)
```

output :-

```
2
3
4
5
```

* The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

Example :-

```
for x in range(2,30,3):
    print(x)
```

output :-

```
2   11   20   29
5   14   23
8   17   26
```

Nested Loops:- (For)

* A nested loop is a loop inside a loop.
* The "inner loop" will be executed one time for each iteration of the "outer loop":

Example:- Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
    for y in fruits:
    print(x, y)
```

Output:-

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Calculating a Running Total:-

```
total = 0
for i in range(5):
    newnumber = int(input("Enter a number"))
    total += newnumber
print("the total is": total)
```

Output:-

```
Enter number 3
Enter number 2
enter number 3
```

Enter number 4

Enter number 3

the total is : 12

Input validation loops:-

```
age = int(input("please enter your age:"))
if age >= 18:
    print("you are eligible to vote")
else:
    print("you are not eligible to vote")
```

Output:-

Please enter your age : 8

you are not eligible to vote

G. Soumya

Program Development cycle:-

* Python's development cycle is dramatically shorter than that of traditional tools. In python, there are no compile or link steps -- Python Programs simply import modules at runtime and use the objects they contain
* Because of this, Python Programs run immediately after changes are made.
* And in cases where dynamic module reloading can be used, it's even possible to change and reload parts of a running Program without stopping it at all.
* The following fig shows python's impact on the development cycle:

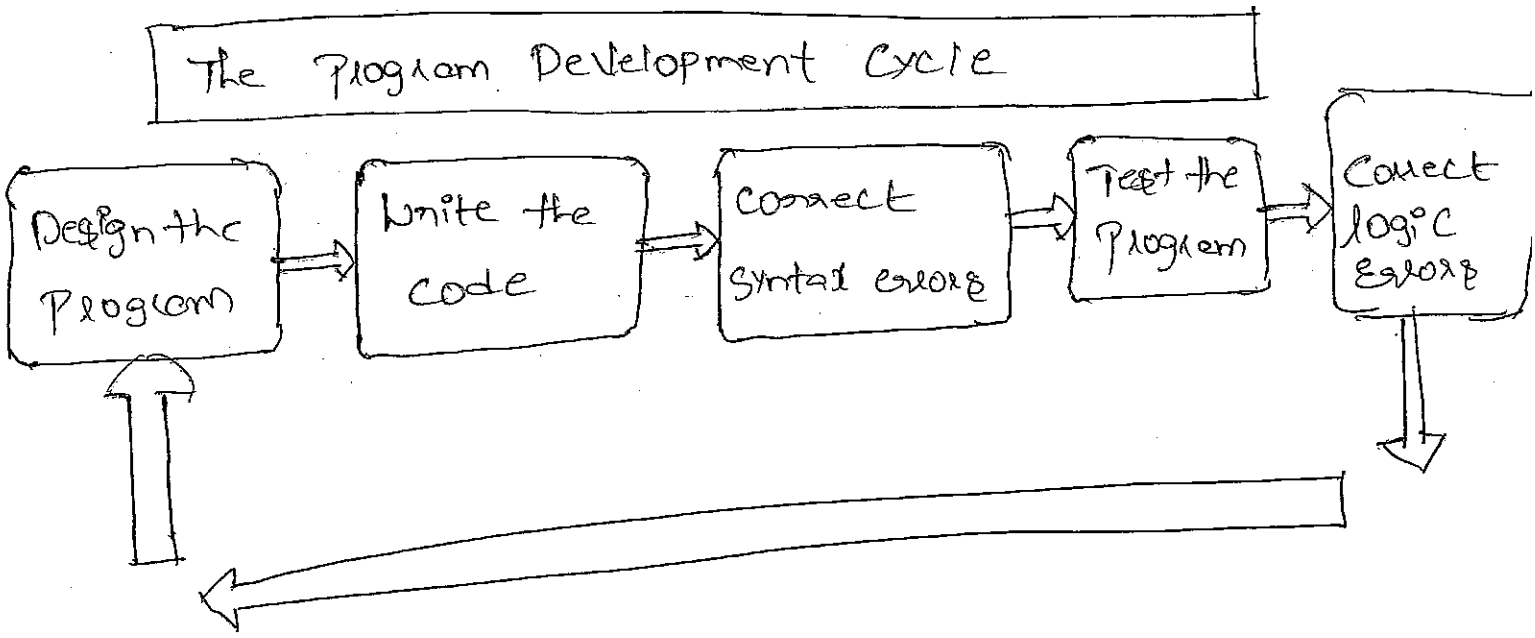


fig: Program development cycle

UNIT-2

Strings and Text files:-

Accessing characters and Substring in strings:-

Accessing characters in python:-

* In python, individual characters of a string can be accessed by using the method of indexing, indexing allows negative address references to access characters from the back of the string e.g -1 refers to the last character, -2 refers to the second last character and so on.

S	O	W	M	Y	A	D	A	T	T	A
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Example:-

```
a = "SowmyaDatta"
Print(a)
Print("\n First character of string is :")
Print ( a[0])
Print("\n Last character of string is :")
Print ( a[-1])
```

Output:-

```
SoumyaDatta
First character of string is :
S
Last character of string is :
a
```

Substring in strings :-

TO access a range of characters in the string, method of slicing is used. Slicing in a string is done by using a slicing operator (colon).

Example:-

```
a = "SoumyaDatta"
Print(a)
Print("In slicing characters from 3-12:")
Print(a[3:12])
Print("In slicing characters between" +
      "3rd and 2nd last character:")
Print(a[3:-2])
```

G. Soumya

Strings and Number Systems :-

In python, strings or numbers can be converted to a number of strings using various inbuilt functions like str(), int(), float() etc)

Example:- Converting python string to an int:

```
a = "30"
print (int(a) + 20)
```

output:-

50

Example2:- Converting a python string to float:

```
a = "10"
print (float(a) + 2.0)
```

output:- 12.0

Example 3:- Converting a python int to a string:

This is achieved by using str() function:

```
a = 100
print (str(a) + "is a 3 digit number")
print (str(a) + "200")
```

output:-

```
100 is a 3 digit number
100200
```

Example 4: converting a python float to a string:

This is achieved by using float() function as shown below:

```
a = 20.0
Print (str(a) + "is now a string")
Print (str(a) + "30.0") # no addition is Performed, Concatenated output
```

Output:-

20.0 is now a string

20.030.0

String Methods Text files:-

Python Provides inbuilt functions for creating, writing and reading files. There are two types of files that can be handled in python, normal text files and binary files (written in binary language, 0's and 1's)

→ Text files:- In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is new line character ('\n') in python by default.

→ Binary files:- In this type of file, there is no terminator for a line and the data is stored after converting it

into machine understandable binary language.

File access modes:-

→ Read only ('r'): Open text file for reading

→ Read and Write ('r+'):- Open the file for reading & writing

→ Write only ('w'):- Open the file for writing

→ Write and Read ('w+'):- Open the file for reading & writing

→ Append only ('a'):- Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

→ Append and Read ('a+'):- Open the file for reading and writing.

Opening a file:- It is done using the open() function.

Syntax:-

File_object = open ("Filename", "Access_Mode")

Example:-

file1 = open ("MyFile.txt", "a")

file2 = open (r"D:\Text\MyFile2.txt", "w+")

Closing a file:- close() function closes the file and frees the memory space acquired by that file. It is used at the time when the file is no longer needed or if it is to be opened in a different file mode.

Example:-

```
#Opening and closing a file "MYfile.txt"
file1 = open("MYFile.txt", "a")
file1.close()
```

Writing a file :- There are two ways to write in a file.

→ write(): Inserts the string str1 in a single line in the text file

```
file_object.write(str1)
```

→ writelines():- For a list of string elements, each string is inserted in the text file. Used to insert multiple strings at a single time.

```
File_object.writelines(L) for L = [str1, str2, str3]
```

Reading from a file:- There are 3 ways to read data from a text file.

→ read()
→ readline()
→ readlines()

G. Sowmya

→ read():- Returns the read bytes in form of a string. Reads n bytes, if no n specified, reads the entire file.

Example:-

File_object.read([n])

→ readline():- Reads a line of the file and returns in form of a string. For specified n, reads at most n bytes. However, does not reads more than one line, even if n exceeds the length of the line.

Example:-

File_object.readline([n])

→ readlines():- Reads all the lines and return them as each line a string element in a list.

Example:-

file_object.readlines()

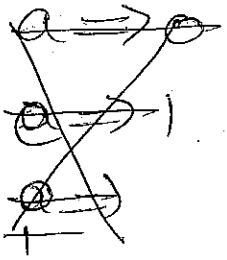
———— X ————

Data Encryption:-

* Encrypt the string according to the given algorithm in python:

→ Given a string's, the task is to encrypt the string in the following way. Let the string be "apple".

- Step 1: Reverse the input: "elppa".
- Step 2: Replace all vowels using the following chart:



a ⇒ 0

e ⇒ 1

i ⇒ 2

o ⇒ 2

u ⇒ 3

Resultant string = "11PP0"

- Step 3:- Add "aca" to the end of the word

Resultant string : "11PP0aca"

Examples:

Input: banana

Output: 0n0n0baca

Input: kareca

Output: 0c0r0kaca

Input: burak

Output: k0r3baca

Approach:-

1. Create a dictionary which stores values so that it can be easily accessed
2. Reverse the string by indexing method
3. Loop through the word to replace vowels.

G. Sowmya

Example Program:-

```
Encrypt = "banana"
dict = {"a":"0", "e":"1",
        "i":"2", "o":"2",
        "u":"3"}
num = Encrypt[::-1]
for i in dict:
    num = num.replace(i, dict[i])
Print(f"{num}aca")
```

Simple Functions:-

Example 1:- Parameterized function

```
def greet(name):
    print ('Hello', name)
greet ('Steve')
greet (123)
```

Output:-

```
Hello Steve
Hello 123
```

Example 2:-

```
def greet(name1, name2, name3):
    print ('Hello', name1, ',', name2, ', and', name3)
greet ('Tony', 'Bony', 'pony')
```

Output:-

```
Hello Tony , Bony, and Pony
```

Example 3:- unknown number of arguments

```
def greet (*names)
    print ('Hello', names[0], ',', names[1], ',', names[3])
greet ('Sita', 'Gita', 'Ketha')
```

Output:-

Hello Sita, Gita, Ketha

Example 4:- Function with keyword Arguments

```
def greet(firstname, lastname):
    Print ('Hello', firstname, lastname)
greet (lastname = 'Ram', firstname = 'Sita')
```

Output:-

Hello Sita Ram

———— X ————