Program – Set of instructions to perform a specific task by computer.

Level of program

1. High Level
2. Low Level – directly interact with hardware

Searching

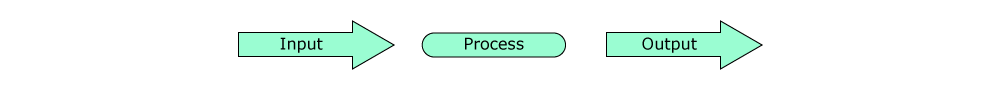
Counting

Optimization – limited resources due to which optimization needed

Sorting

Decision

How to solve problem

The solution to any problem involves the below three aspects:

Algorithm & its Representation

In mathematics and computer science, an algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation.

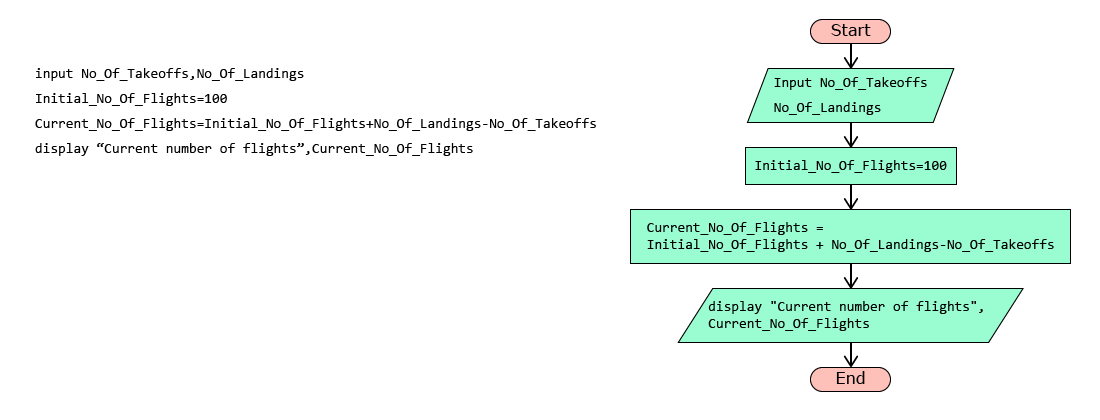
Representation of an algorithm

Algorithm should be represented from our mental thoughts into a form which others can understand. There are primarily two ways of representing an algorithm:

* Flow chart: Diagrammatic way of representing the algorithm.
* Pseudo-code: Representation of the algorithm in a way that is in between a program and normal English.

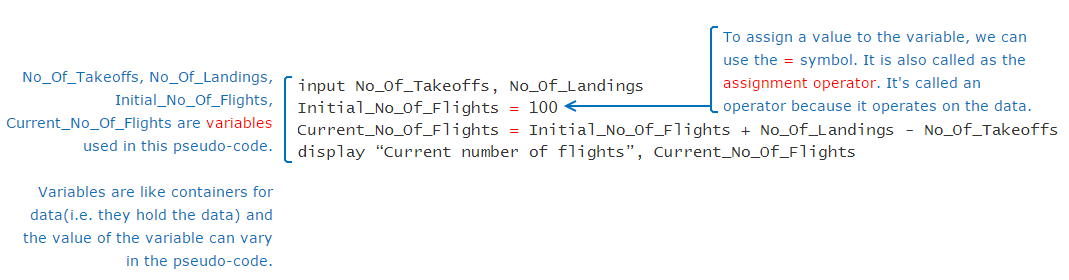
For example, here is a pseudo-code and a flowchart.

**Pseudo-code:**                                                                                                                                             **FlowChart:**



Pseudo-code cannot be executed by a computer. It is just a representation of an algorithm for us to understand.

Variables & Operators



Variable - Container for any value

Like assignment operator, there are other operators also which can be used to perform various operations.  
Arithmetic operators: Used for performing arithmetic operations

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **+** | Addition |
| **-** | Subtraction |
| **\*** | Multiplication |
| **/** | Division |
| **%** | Modulus |

Relational operators: Also known as comparison operators, are used to compare values. Result of a relational expression is always either true or false.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **==** | Equal to |
| **<** | Less than |
| **>** | Greater than |
| **<=** | Less than or equal to |
| **>=** | Greater than or equal to |
| **!=** | Not equal to |

Logical operators are used to combine one or more relational expressions.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| AND | Result will be true, if both the expressions are true. If any one or both the expressions are false, the result will be false |
| OR | Result will be true, even if one of the expression is true. If both the expressions are false, the result will be false |
| NOT | If the expression is true, result will be false and vice versa |

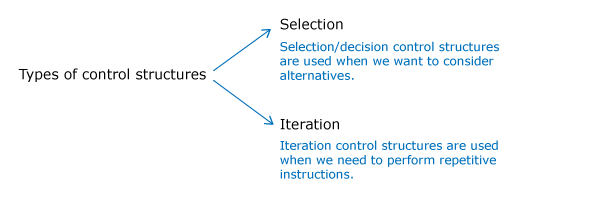
If A and B are two relational expressions, say A = (Num1>2000), B= (Num2>100), the result of combining A and B using logical operator is based on the result of A and B as shown below:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A AND B** |
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

Decision Constructs

Flow of control

In a pseudo-code, typically the instructions are performed one by one or line by line. But there may be situations when all the statements in a pseudo-code are not performed. Parts of the pseudo-code which change the flow of instructions or in other word, change the flow of control are called as control structures.

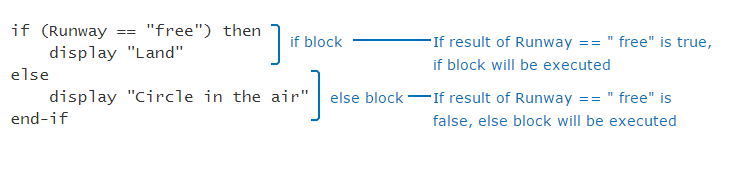


Selection using if statement

ATC takes lot of decisions as part of its air traffic control operations.

For example, if a flight is approaching the runway, ATC has to check if the runway is free. If the runway is not free, then the flight should not land immediately. It should circle in the air and wait for further instructions from the ATC.

Such decision making process can be conveniently represented in a pseudo-code using an if statement.



Sometimes, ATC may have more than one alternatives for a given situation. For example, if the runway is free, the flight can land. But if the flight has less fuel, then it should be allowed an emergency landing. Otherwise, it should circle in the air.

if (Runway=="free") then

   display "Land"

else if (Fuel\_Status=="low") then

   display "Emergency landing"

else

   display "Circle in the air"

end-if

Iteration Constructs

Immigration check needs to be done for all the passengers in the flight. Suppose the flight had only 5 passengers, the pseudo-code can be written as follows:

display "Flight has landed"

display "Proceed for Immigration Check"

Passenger\_Count=1

display "Immigration check done for passenger,", Passenger\_Count

Passenger\_Count=Passenger\_Count+1

display "Immigration check done for passenger,", Passenger\_Count

Passenger\_Count=Passenger\_Count+1

display "Immigration check done for passenger,", Passenger\_Count

Passenger\_Count=Passenger\_Count+1

display "Immigration check done for passenger,", Passenger\_Count

Passenger\_Count=Passenger\_Count+1

display "Immigration check done for passenger,", Passenger\_Count

Sometimes, we may want to execute some statements specific (known) number of times as in the case of immigration check. In such cases we can use a **for loop**as shown below.

No\_Of\_Passengers=5

for(Passenger\_Count=1,Passenger\_Count<=No\_Of\_Passengers,Passenger\_Count=Passenger\_Count+1)

     display "Immigration check done for passenger,", Passenger\_Count

end-for

When we want to repeatedly execute a statement as long as a condition is met, we can use the iteration statement called as **while loop**.

display "The flight has landed"

display "Immigration check done"

display " Collect the baggage from the conveyor belt"

Baggage\_Count=150

while(Baggage\_Count>0) do

    input No\_Of\_Baggage\_Picked

    Baggage\_Count=Baggage\_Count-No\_Of\_Baggage\_Picked

end-while

Go through the below two pseudo-codes and guess the output.

|  |  |
| --- | --- |
| **Pseudo-code 1 :** | **Pseudo-code 2 :** |
| 1. Counter=5 2. while(Counter>=5) do 3. display Counter 4. Counter = Counter + 1 5. end-while | 1. input Limit 2. for(Counter=5, Counter<=Limit, Counter=Counter-1) 3. display Counter 4. end-for   Assume that the input value provided to variable Limit is |

Beware of infinite loops. The logic that we are writing in loops should ensure that the loop will terminate in finite number of iterations.

What is the outcome of the following pseudo-code?

input Counter

while(Counter<5) do

  Counter=Counter+1

  display Counter

end-while

Assume that the input value provided to variable, Counter is 1.

2,3,4,5

2,3,4

1,2,3,4

1,2,3,4,5

Programming languages

Pseudo-code helped us to represent the algorithms and learn few basics of programming. But to instruct the computer we need to write a program in a programming language. There are many languages available in which we can write our programs.

Different languages are created for different purposes. This involves trade-offs. For example a large flight can carry many passengers, but also consumes lot of fuel. Similarly different languages have different advantages.

Datatypes

Data

We know that a program works on data or values. Some examples of values are 1, 1.0 and "Hello".For a program to use a data, it must be stored in a memory location. The values are stored in binary form in computer memory.

How do we know how many bits are required to represent a value or how much space it needs in memory?

This is determined by the data type of the value and the programming language. e.g. number, string etc. Thus every value will have a type.

Datatypes

Data Type determines the operations that can be performed on a value. For e.g. we can perform operations like addition, multiplication, division, subtraction etc. on numerical data types. We can concatenate, convert case, extract substring etc. on string data types.

1. print(1 + 2)
2. print("Hello " + "World")
3. print(True and False)
4. print(4.0 / 2.0)

The above program uses 8 values of different data types.

A program may have data belonging to different types. Common data types used in programming are:

|  |  |  |
| --- | --- | --- |
| **Category** | **Data Type** | **Example** |
| Numeric | int | 123 |
| long | 1237126381763817 |
| Numeric with decimal point | float | 123.45 |
| double | 123123.32345324 |
| Alphanumeric | char | A |
| String | hello |
| Boolean | boolean | True, False |

Python programming language supports the following datatypes:

|  |  |
| --- | --- |
| **Category** | **Python** |
| Numeric | int |
| long |
| complex |
| Numeric with decimal point | float |
| Alphanumeric | String |
| Boolean | boolean |

### **Problem Statement**

In Python, the data type of a value can be identified by using **type**(value).

Try out the below program and observe the results.

print(type(3))

print(type("Hello World"))

print(type(False))

print(type(2.0))

Output:

<class 'int'>  
<class 'str'>  
<class 'bool'>  
<class 'float'>

Variables



If everything is a value, then what is a variable?

A variable is a name that is assigned to a value. It is done so that we can refer to that value at some later point in the program.

Let us see the usage of values and variables in a program.

* During input, we receive values and assign them to variables.
* During processing, we perform operations on values and variables to generate more values. These computed values are also assigned to variables.
* During output, we display the computed values in variables to the end users.

Let’s have a look at the python program to display the number of landings and number of takeoffs in an airport:

Note: In Python, print() can be used to display output in the console

### **Problem Statement**

Try out the below program. This is the program for a pseudo-code which we had discussed earlier.  
Change the value of no\_of\_landings, no\_of\_takeoffs and initial\_no\_of\_flights, execute and observe the output.

no\_of\_landings=356

no\_of\_takeoffs=245

initial\_no\_of\_flights=100

current\_no\_of\_flights=initial\_no\_of\_flights+no\_of\_landings-no\_of\_takeoffs

print("Current number of flights:",current\_no\_of\_flights)

Output:

Current number of flights: 211

Static typed language fast – java

Dynamic typed language slow - python

Variables & Datatypes

Let’s see how can we associate a data type with a variable.

|  |
| --- |
| **Python** |
| num=100  msg="Hello" |

Why do you think the data type is not mentioned in Python?

Static and dynamic typing

Languages like Python are dynamically typed whereas C,Go etc are statically typed.

**Dynamic Typing**is a technique in some languages where depending on how a value is used, the data type of the variable is dynamically and automatically assigned. Consider the below code in Python,

num=65 #Line 1

num="A" #Line 2

In Line 1, variable num is considered to be of type int and in Line 2, its type is reassigned to String.

**Static Typing** is used in some languages where the data type has to be declared before a variable is used. Consider the below code in Go,

var num int=65; //Line 1

num="A"; //Line 2

var name string = "A"; //Line3

Here, Line 1 is a valid statement which declares a variable num of type int. But Line 2 is invalid as we cannot assign a string value to variable num which is already declared to be of type int. Line 3 is a valid statement where name is declared and used as a string.

Reserved words

Any name can be given to a variable however, we cannot use some of the built-in keywords of the language. These keywords are known as **reserved words**. Some of the reserved words in Python are:

|  |  |
| --- | --- |
| **Python** | if, else, for, while, def, print, raise, try, except |

Storing data

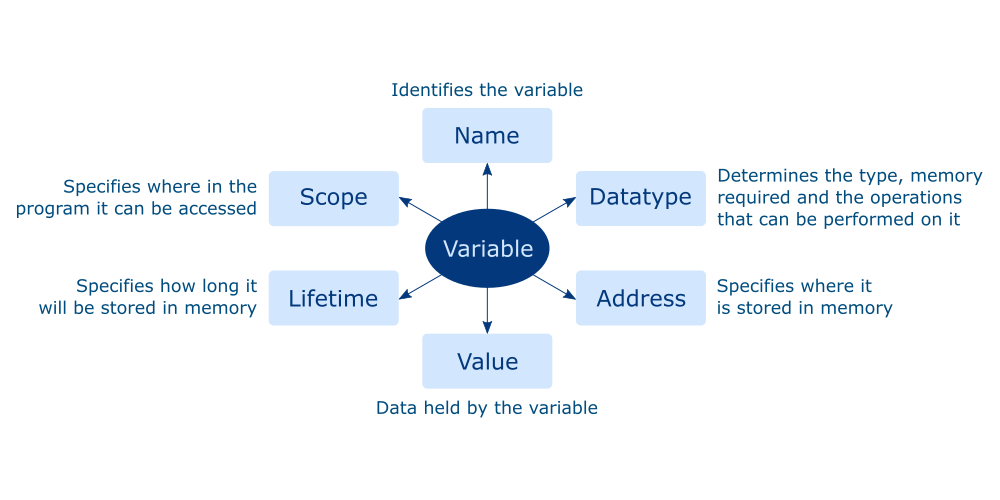
If you put a small product in a big box, you may end up wasting space. Similarly if you put a big product in a small box you may end up damaging the product.

Just like product is placed in a box, data occupies memory. Some data need more memory whereas some other data require less memory based on the data type.

In Python language, automatically creates just the right memory needed – neither less nor more.

Variable and its dimensions

We have seen that a variable will have a name, value, type and it will occupy memory. Apart from these, it has two more dimensions – scope and lifetime. Thus we can say that any variable will have the following six dimensions.



Operators

As we discussed in pseudo-code, operators help to perform an operation.

Some of the most common operators used in Python are listed below.

|  |  |
| --- | --- |
| **Common Operators** | **Python** |
| Arithmetic Operators | +,-,\*,/, %,// |
| Relational Operators | ==,!=,>,<,>=,<= |
| Assignment Operators | =,+=,-=,\*=,/=,%= |
| Logical Operators | and,or,not |

Note: In Python, // indicates integer division.

         Example: 11//2=5

True or False

The result of a relational or logical expression is always a boolean (true or false). Apart from the boolean values, other values can also be used to represent a true or false value.

For example, in python, value zero is considered to be equivalent to false.

The common false values in Python are given below. Any other value is considered to be true.

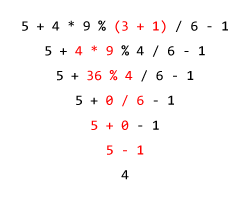
|  |
| --- |
| **Python** |
| None |
| False |
| 0 |
| ''/"" (Empty string - two single quotes/double quotes) |

Precedence of operators

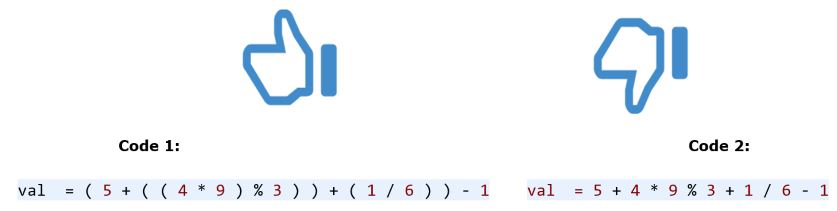
What do you think is the output of **5+4\*9%(3+1)/6-1**?  
How do you think the result of this expression is computed?

It is done based on the precedence of the operators. Precedence of an operator can be identified based on the rule - BODMAS. Brackets followed by Orders (Powers, Roots), followed by modulo, Division and Multiplication, followed by Addition and Subtraction.

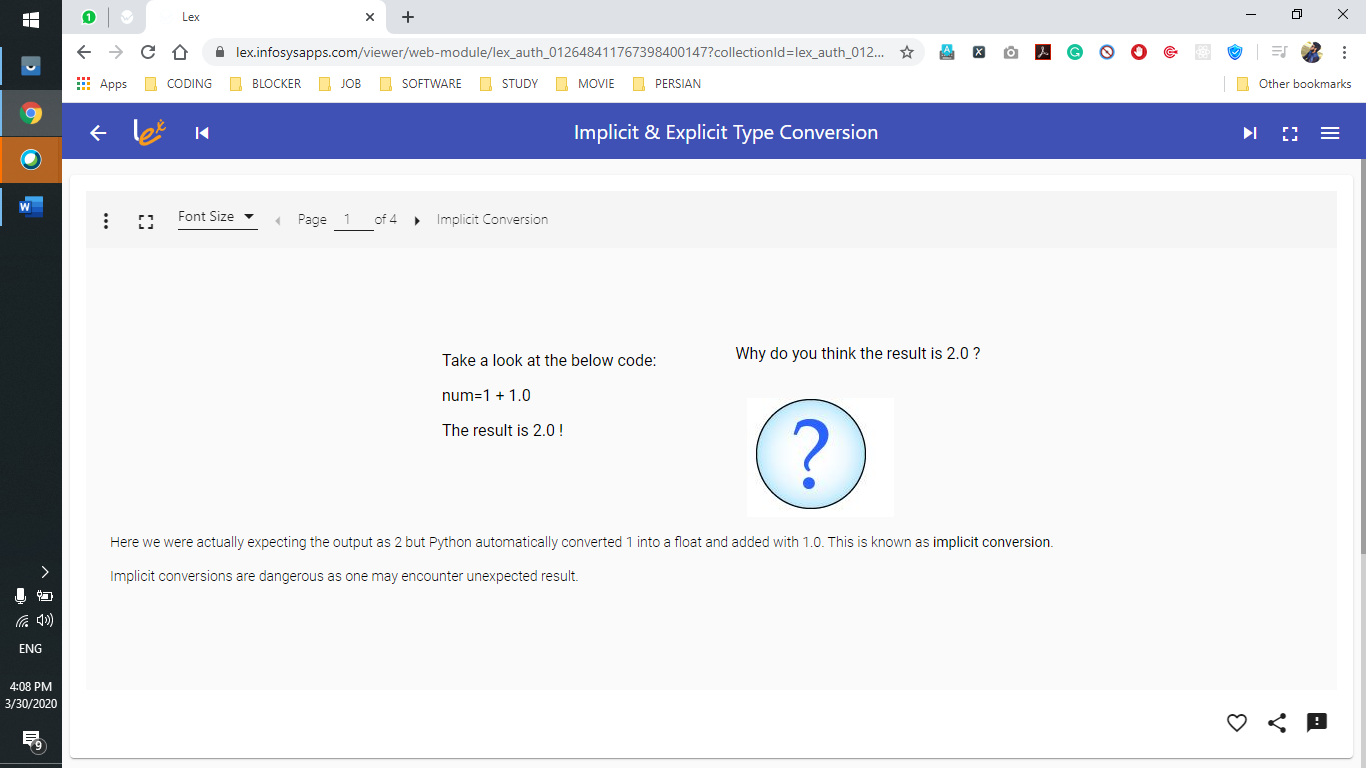
1. Brackets have the highest precedence followed by orders.
2. Modulo, Division and Multiplication have the same precedence. Hence if all appear in an expression, they are evaluated from Left to Right.
3. Addition and Subtraction have the same precedence. Hence if both appear in an expression, they are evaluated from Left to Right.

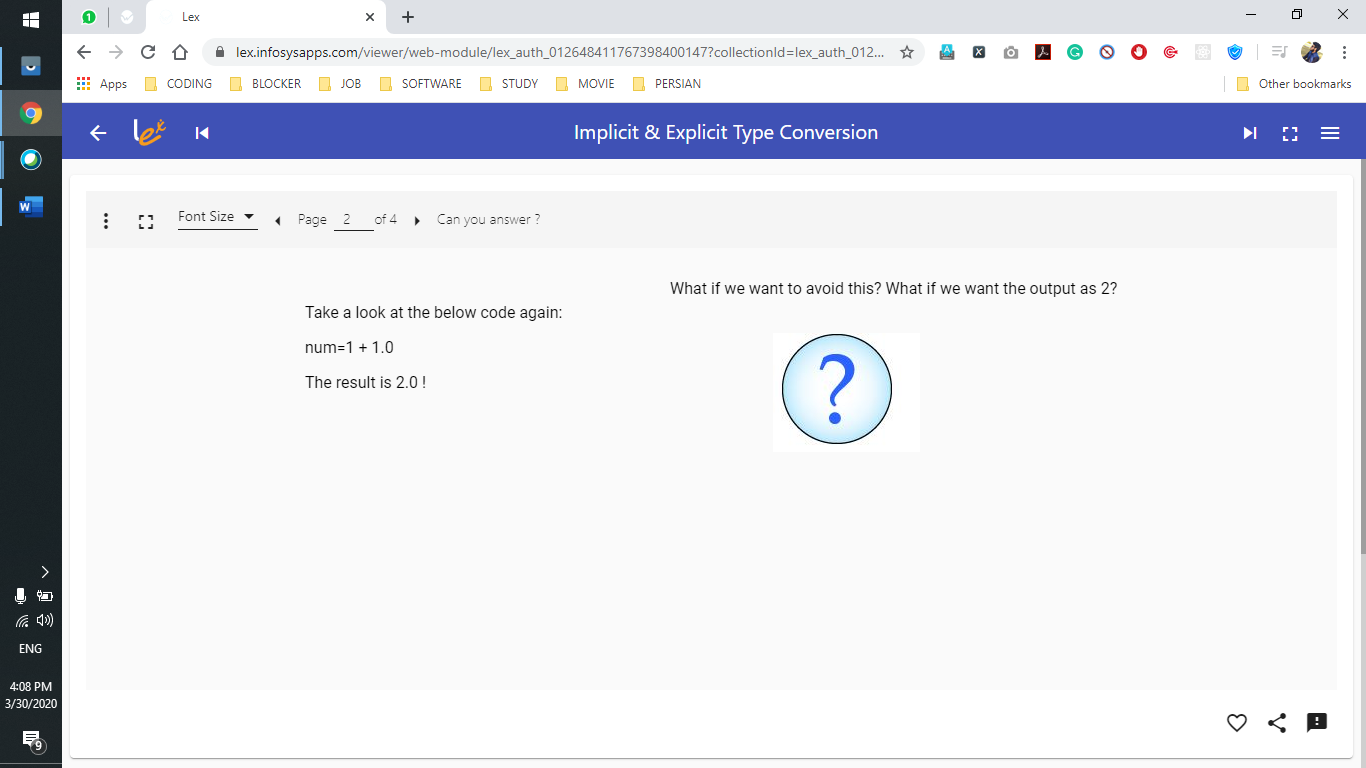


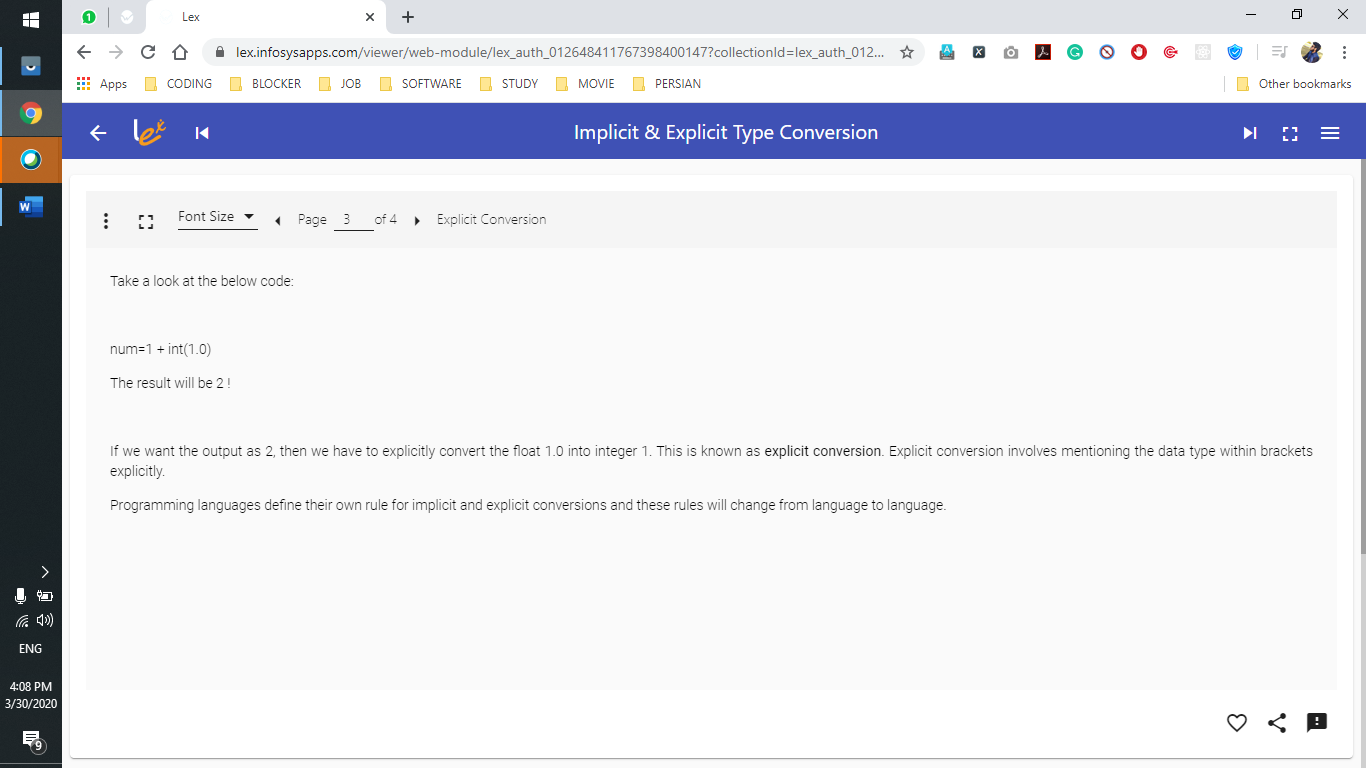
Though its not compulsory to have brackets to denote the precedence of operators in an expression, it is always preferred to have brackets as it makes the expression readable and brings in a lot more clarity.

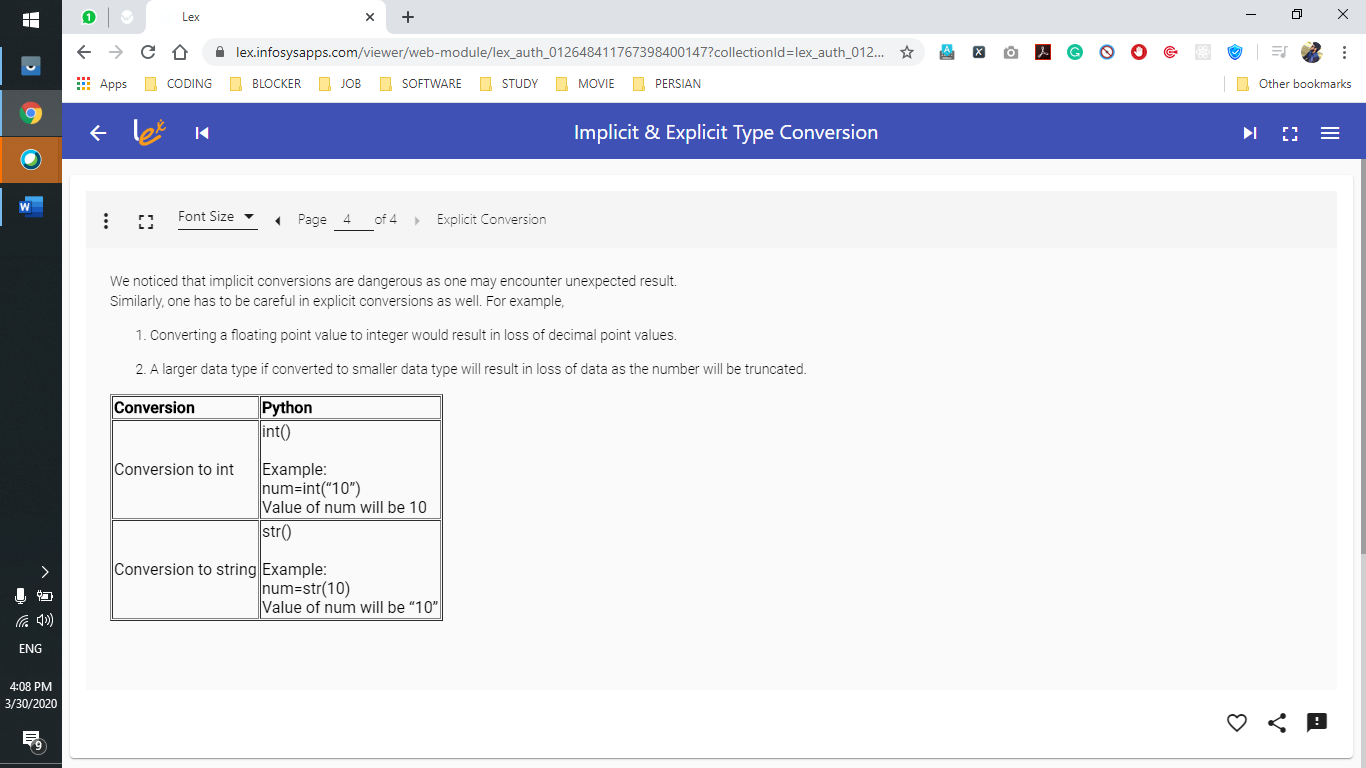


Implicit & Explicit Type Conversion









DAY 2 March 31st

String

In a program, not all values will be numerical. We will also have alphabetical or alpha numerical values. Such values are called strings.  
**Example:** “Hello World”, pancard number - “AABGT6715H” , name – “Arjun”, ticket number – “BA1045”   
Each value in a string is called a **character**. Just like list elements, we can access the characters in a string based on its index position.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| String | "AABGT6715H" | | | | | | | | | |
| Character | A | A | B | G | T | 6 | 7 | 1 | 5 | H |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

In Python, string is a data type and anything enclosed in a single quote or double quote is considered to be a string. All the remaining operations are similar to lists. But like tuple, strings are also **IMMUTABLE.**

# String is another type of collection

# String is also immutable

# String Follow same concept of indexing

# String can be added

# String Declaration

# string\_one=""

# string\_two=''

# String Addition

string\_one="alpha"

string\_two='beta'

string\_one=string\_one+'delta'

print(string\_one)

#  when we add anything to a string new value is allocated to the string.

Try out - String  
#Creating a string

pancard\_number="AABGT6715H"

#Length of the string

print("Length of the PAN card number:", len(pancard\_number))

#Concatenating two strings

name1 ="PAN "

name2="card"

name=name1+name2

print(name)

print("Iterating the string using range()")

for index in range(0,len(pancard\_number)):

    print(pancard\_number[index])

print("Iterating the string using keyword in")

for value in pancard\_number:

    print(value)

print("Searching for a character in string")

if "Z" in pancard\_number:

    print("Character present")

else:

    print("Character is not present")

#Slicing a string

print("The numbers in the PAN card number:", pancard\_number[5:9])

print("Last but one 3 characters in the PAN card:",pancard\_number[-4:-1])

# pancard\_number[2]="A" #This line will result in an error, i.e., string is immutable

print(pancard\_number)

Exercise on String - Level 2

### **Problem Statement**

Write a python program to generate the ticket numbers for specified number of passengers traveling in a flight as per the details mentioned below:  
The ticket number should be generated as **airline:src:dest:number**  
where

1. Consider AI as the value for airline
2. src and dest should be the first three characters of the source and destination cities.
3. number should be auto-generated starting from 101

The program should return the list of ticket numbers of last five passengers.  
**Note:** If passenger count is less than 5, return the list of all generated ticket numbers.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| airline = AI source = Bangalore destination = London no\_of\_passengers = 10 | ['AI:Ban:Lon:106', 'AI:Ban:Lon:107', 'AI:Ban:Lon:108', 'AI:Ban:Lon:109', 'AI:Ban:Lon:110'] |
| airline = BA source = Australia destination = France no\_of\_passengers = 2 | ['BA:Aus:Fra:101', 'BA:Aus:Fra:102'] |

def generate\_ticket(airline,source,destination,no\_of\_passengers):

    ticket\_no=100

    ticket\_number\_list=[]

    while no\_of\_passengers>0:

        temp\_ticket=airline+':'+source[:3]+':'+destination[:3]+':'+str((ticket\_no+1))

        ticket\_no+=1

        ticket\_number\_list.append(temp\_ticket)

        no\_of\_passengers-=1

    return ticket\_number\_list[-5:]

#Provide different values for airline,source,destination,no\_of\_passengers and test your program

print(generate\_ticket("AI","Bangalore","London",10))

Try out - String Functions

String data type in Python has many inbuilt functions which make it easier to work with strings.

Consider the string, name="Raghav".

|  |  |  |
| --- | --- | --- |
| **Function** | **Output** | **Explanation** |
| name.count("a") | 2 | Returns the count of a given set of characters. Returns 0 if not found |
| name.replace("a","A") | RAghAv | Returns a new string by replacing a set of characters with another set of characters. It does not modify the original string |
| name.find("a") | 1 | Returns the first index position of a given set of characters |
| name.startswith("Ra") | True | Checks if a string starts with a specific set of characters, returns true or false accordingly. |
| name.endswith("ha") | False | Checks if a string ends with a specific set of characters, returns true or false accordingly. |
| name.isdigit() | False | Checks if all the characters in the string are numbers, returns true or false accordingly. |
| name.upper() | RAGHAV | Converts the lowercase letters in string to uppercase |
| name.lower() | raghav | Converts the uppercase letters in string to lowercase |
| name.split("a") | ['R', 'gh', 'v'] | Splits string according to delimiter and returns the list of substring. Space is considered as the default delimiter. |

Try out the code which uses the String built-in functions and observe the results.

# String Funstions

boarding\_call="Good Evening, this is the final call to AI passengers for the flight AI 466 which is planned to take off at 8.40A.M."

if(boarding\_call.startswith("Good Evening")):

    print(boarding\_call.replace("Good Evening","Good Morning"))

if(boarding\_call.find("AI"))>=0:

    print("Welcome to Air India.")

if(boarding\_call.endswith("A.M.")):

    print("Passengers are requested to have their breakfast.")

a=boarding\_call.split(" ")

for i in a:

    if(i.isdigit()):

        print("Flight Number is specified to the passengers.")

print("Total number of times flight service name is specified in the boarding call:",boarding\_call.count("AI"))

message="Thank you all..Have a nice journey!"

print(message.upper())

print(message.lower())

OUTPUT

Good Morning, this is the final call to AI passengers for the flight AI 466 which is planned to take off at 8.40A.M.

Welcome to Air India.

Passengers are requested to have their breakfast.

Flight Number is specified to the passengers.

Total number of times flight service name is specified in the boarding call: 2

THANK YOU ALL..HAVE A NICE JOURNEY!

thank you all..have a nice journey!

Choosing Between List, Tuple and String

### **Problem Statement**

* Represent the data in the table below in Python such that the data in a row cannot be changed but rows can be shifted up or down in the table
* Strings are used when you want a sequence of characters to act as a unit of information.
* Lists and tuples when you want to have a sequence of individual elements.

#string4.py

row1 = (101,"Dallas",3.5)

row2 = (102,"Atlanta",5.6)

row3 = (103,"Tokyo",9.8)

table = [row1,row2,row3]

print(table[0])

print(table[1])

print(table[2])

Quiz – String

### **Q1 of 3**

### Consider the following list of pan card numbers: pancard\_list=["AABGT6715H", "UFFAC4352T", "IFSBD9163K", "JOOEC1225H","RWXAFE187B"]

What is the output of the below two print statements?

print(pancard\_list[3][6], end=" ")

print(pancard\_list[4][3:])

2 AFE187B

### **Q2 of 3**

What is the output of the code given below?

message="welcome to Mysore"

word=message[-7:]

if(word=="mysore"):

    print("got it")

else:

    message=message[3:14]

    print(message)

come to Mys

### **Q3 of 3**

What is the output of the below code?

song="JINGLE Bells jingle Bells Jingle All The Way"

song.upper()

song\_words=song.split()

count=0

for word in song\_words:

if(word.startswith("jingle")):

count=count+1

print(count)

1

Exercise on String Functions - Level 1

### **Problem Statement**

Write a python program which displays the count of the names that matches a given pattern from a list of names provided.

Consider the pattern characters to be:

1. "\_ at" where "\_" can be one occurrence of any character

2. "%at%" where "%" can have zero or any number of occurrences of a character

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| [Hat, Cat, Rabbit, Matter] | \_at -> 2 %at% -> 3 |

def count\_names(name\_list):

    count1=0

    count2=0

    #start writing your code here

    for i in name\_list:

        if(((i.find("at"))==1) and len(i)==3):

            count1+=1

        if(i.find("at"))>=0:

            count2+=1

    print("\_at -> ",count1)

    print("%at% -> ",count2)

name\_list=["Hat","Cat","rabbit","matter"]

# name\_list=['Rat', 'saturday']

count\_names(name\_list)

Set