Date May 10, 2021

classmate
Date
Page

WEEK 2

#LECTURE 2

VARIABLE'S NAME

→ As a programmer, you will have to write variables daily.

→ So, the name of the variable also plays a very vital role in coding. (for readability & understanding)

→ Keeping variables names like a, b, x, y are not much usable or readable, because, for example, if anyone else is reading your code, that person should know what that variable stands for.

→ So, prefer keeping the names of variables related to whatever you are storing in it.

For example,

```
a = 'Gagneet'
b = 18
```

Now here I wanted to store my name and age, but it is not so useful to write the names of variables as 'a' and 'b' for name & age resp. So, it should be,

```
name = 'Gagneet'
age = 18
```

- It is not compulsory, but it makes your program more readable and understandable to someone else.

USE COMMENTS

- Again, for the same purpose, i.e, making your program understandable to other people, you can use comments.

- Use '#' to write comments.

For Example,

```
name = 'Gagneet'
# this variable stores my name
age = 18
# this variable stores my age
```

- Comments are not executed by compiler.
- They are used only for your code understanding purpose.
- They will not be visible in the output screen.

May 11, 2021

LECTURE 3

DYNAMIC TYPING

→ Python is a dynamically typed language.

→ The Python Interpreter does type checking only as code runs.

→ The type of a variable is allowed to change over its lifetime.

EXAMPLE :

```
thing = 'Hello World'
print ( type (thing))
```

Output : <class 'str'>

```
thing = 12
print ( type (thing))
```

Output: <class 'int'>

→ We changed the 'type' of the variable 'thing' two times.
It has been dynamically typed, i.e, changed its datatype in the same program.

→ In other languages such as C++ & java, static typing check occurs. They are statically typed languages. However, with type casting, dynamic typing is allowed in such languages.

LECTURE 4

RULES TO WRITE A VAR. NAME

→ A variable name should start with a letter or an underscore (_)

→ A variable name can't start with a number.

→ A variable name is case sensitive.

→ A variable name should only contain underscores and alphanumeric characters.

→ A variable name should not be a keyword in python.

Examples of some keywords :

and , or , not , for , if , else , while .... and so on .

MULTIPLE ASSIGNMENT

→ Python allows you to assign values to multiple variables in one line.

Example CODE:

```
x,y = 1,2
print (x, y)
```

OUTPUT : 1 2

CODE :
```
x = y = z = 8
print (x, y, z)
```

OUTPUT : 8 8 8

→ If the no. of variables on the left and the no. of values on the right do not match, a [Value Error] will occur, but you can assign the rest as a list by appending * to the variable name.

CODE :
```
# a, b = 100, 200, 300
# Value Error → too many values to unpack

# a, b, c = 100, 200
# Value Error → not enough values to unpack
```

```
a, *b = 100, 200, 300

print (a)                # 100
print (b)                # [200, 300]
print (type (a), type (b))
```

OUTPUT :
```
100
[200, 300]
<class 'int'> <class 'list'>
```

```
*a, b = 100, 200, 300
print (a)
print ( type (a))
print (b)
print (type (b))
```

OUTPUT :
```
[100, 200]
<class 'list'>
300
<class 'int'>
```

DELETING A VARIABLE

Using del() function, we can delete any object or variable in python.

For example:

CODE :
```
x = 3
print (x)
del (x)
print (x)
```

OUTPUT :
```
3
Trackback (most recent call last)
File "name", line4
print(x)
Name Error: name 'x' is not defined
```

→ The 'Name Error' occurs because you have deleted the variable 'x'.

SHORT HAND OPERATORS

PYTHON ASSIGNMENT OPERATORS

OPERATOR	EXAMPLE	SAME AS
+=	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

MORE OPERATORS IN PYTHON

MEMBERSHIP OPERATORS

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object.	x not in y

Code :

```
sentence = 'I am Gagneet Kaur'
print ('Kaur' in sentence)

sent2 = 'I am a girl'
print ('girl' not in sent2)
```

Output :

```
True
False
```

CHAINING IN COMPARISON OPER.

(1) Comparisons yield boolean values : True or False.

(2) Comparisons can be chained arbitrarily.
For example,

$(x < y <= z)$ is equivalent to $(x < y$ and $y <= z)$

except that y is evaluated only once.
(but in both cases z is not evaluated at all when $x < y$ is found to be false)

(3) Formally, if a, b, c, ..., y, z are expressions and op1, op2, ..., opN are comparison operators, then,

a op1 b op2 c ..... y opN z is equivalent to
a op1 b and b op2 c and ... y opN z,

except that each expression is evaluated at most once.

(4) Also,

$$\boxed{a \text{ op1 } b \text{ op2 } c}$$

doesn't imply any kind of comparison between a and c, so

$$\boxed{a < b > c}$$

is perfectly legal.

CODE :

```
x = 5
print (1 < x < 10)
print (10 < x < 20)
print (x < 10 < x*10 < 100)
```

```
print (10 > x <= 9)
print (5 == x > 4)
```

Output: True
False
True
True
True

#LECTURE 5

ESCAPE CHARACTER

Escape characters are characters that are generally used to perform certain tasks and their usage in code directs the compiler to take a suitable action mapped to that character.

EXAMPLE:

'\n' → Leaves a line
'\t' → Leaves a space

~~Also~~, ~~for~~

→ An escape character is a backslash '\' followed by the character you want to insert.
→ To insert characters that are illegal in a string, use an escape character.
→ An example of an illegal character is a double quote inside a string that surrounded by double quotes.

Example: You will get an error if you use double quotes inside a string that is surrounded by double quotes.

CODE:
```
txt = "We are the so-called "Vikings" from north"
print (txt)
```

OUTPUT: syntaxerror : invalid syntax

To fix this problem, use the escape character '\':

CODE:
```
txt = "We are the so called \"Vikings\" from north."
print (txt)
```

OUTPUT: We are the so called "Vikings" from north.

Other Escape Characters in Python

CODE	RESULT	DESCRIPTION
print ("\'")	'	Single quote
print ("\\")	\	Backslash
print ('\n')	new line	New line
print ('\t')	tab space	Tab

DIFFERENT TYPES OF QUOTES

→ Single Quote : ' '
Double Quote : " "
Triple Quotes : ''' '''

→ Single and double quotes functions similarly. They are used for printing single line strings.

→ But, what if you want to print multi line strings,

USE TRIPLE QUOTES

CODE:
```
z = """ one
        two
        three """
print (z)
```

OUTPUT:
```
one
two
three
```

→ Also, to print multi line comments, we use triple quotes. (for single line → #)

CODE:
```
""" this is
    multi line
    comment """
z = 'Multi line'
print (z)
```

OUTPUT: Multi line

Date May 13, 2021

classmate
Date
Page

#LECTURE 6

String Methods

(1) **Lower Case** lower( )

It converts a string into lower case.

CODE:
```
x = 'pyTHon'
print (x.lower())
```
OUTPUT : python

(2) **Upper Case** upper( )

It converts a string into upper case.

CODE:
```
x = 'pyTHon'
print (x.upper())
```
OUTPUT: PYTHON

(3) **Capitalize** capitalize( )

It converts the first character to upper case.

CODE:
```
x = 'i love python'
print (x.capitalize())
```
OUTPUT : I love python

(4) **Title** title( )

It converts the first character of each word into upper case.

CODE: x = 'i love python'
print (x.title())

OUTPUT: I Love Python

(5) Swapcase swapcase()

It swaps the cases, lower case becomes upper case and vice versa.

CODE: x = 'pyTHon'
print (x.swapcase())

OUTPUT: PYthON

(6) Checking Lower Case islower()

It returns True if all characters in the string are lower case.

CODE 1: x = 'python'
print (x.islower())

OUTPUT: True

CODE 2: x = 'Python'
print (x.islower())

OUTPUT: False

(7) Checking Upper Case isupper()

It returns True if all characters in the string are

upper case.

CODE 1: x = 'PYTHON'
print (x.isupper())
OUTPUT: True

CODE 2: x = 'PYTHON'
print (x.isupper())
OUTPUT: False

(8) Checking Title istitle()

It returns True if the string follows the rules of a title.

CODE 1: x = 'Python String Methods'
print (x.istitle())
OUTPUT: True

CODE 2: x = 'Python string methods'
print (x.istitle())
OUTPUT: False

(9) Checking Numbers isdigit()

It returns True if all characters in the string are digits.

CODE 1: x = '123'
print (x.isdigit())
OUTPUT: True

CODE 2: x = '123abc'
print (x.isdigit())
OUTPUT: False

(10) Checking Alphabets isalpha()

It returns True if all characters in string are in alphabets.

CODE 1 :
```
x = 'abc'
print (x.isalpha())
```
OUTPUT : True

CODE 2 :
```
x = 'abc123'
print (x.isalpha())
```
OUTPUT : False

(11) Checking Alphanumeric Characters ~~isalp~~ isalnum()

It returns True if all the ~~to~~ characters in the string are alpha-numeric.

CODE 1 :
```
x = 'abc123'
print (x.isalnum())
```
OUTPUT : True

CODE 2 :
```
x = 'abc123@#'
print (x.isalnum())
```
OUTPUT : False

(12) Trimming Strings

METHOD	DESCRIPTION	CODE x = '---Python---'	OUTPUT
strip()	Returns a trimmed version of the string	~~print(x.strip())~~ print(x.strip('-'))	Python
lstrip()	Returns a left trim version of the string	~~print(x.lstrip())~~ print(x.lstrip('-'))	Python---
rstrip()	Returns a right trim version of the string	~~print(x.rstrip())~~ print(x.rstrip('-'))	---Python

(13) Checking the Start Value of string

startswith( )

It returns True if the string starts with the specified value.

CODE 1 :
```
x = 'Python'
print(x.startswith('P'))
```
OUTPUT : True

CODE 2 :
```
x = 'Python'
print(x.startswith('p'))
```
OUTPUT : False

(14) Checking the EndValue of String

endswith( )

It returns True if the string ends with a specified value.

CODE 1:
```
x = 'Python'
# print (x.start
print (x.endswith('n'))
```
OUTPUT: True

CODE 2:
```
x = 'Python'
print (x.endswith('N'))
```
OUTPUT: False

(15) Count

count( )

It returns the number of times a specified value occurs in a string.

CODE 1:
```
x = 'Python string methods'
print (x.count('t'))
print (x.count('s'))
```
OUTPUT: 3
2

(16) Index

index( )

It searches the string for a specified value and returns the position of where it was found.

CODE:
```
x = 'Python'
print (x.index('t'))
```
OUTPUT: 2

(17) **Replace** replace()

It returns a string where a specified value is replaced with a specified value.

CODE:
```
x = 'Python string methods'
x = x.replace('s', 'S')
x = x.replace('m', 'M')
print(x)
```

OUTPUT: Python String Methods

Date May 16, 2021

#LECTURE 7

Caeser Cipher

→ Let us just start from basic code and then move on to caeser cipher!

#1 CODE

```
alpha = 'abcdefgihjklmnopqrstuvxyz'

i = 0
print ( alpha[i])      #indexing
print ( alpha[i+1])
print ( alpha[i+2])
```

OUTPUT

```
a
b
c
```

→ Here we simply used indexing.
Now, in a situation, there rises an error.
The index of alpha ranges from 0 - 25 because there are in total 26 letters.

Let us take a look at that situation:

CODE

```
alpha = 'abcdefghijklmnopqrstuvwxyz'
i = 25
print ( alpha [i])          # it will print letter with index 25, i.e, z
print ( alpha [i+1])        # it will produce error
```

OUTPUT

```
z
Traceback (most recent call last):
    File "name" , line 4 , in <module>
        print (alpha [i+1])

IndexError : string index out of range
```

→ Now,

In this case we get an index error.
What should we do now ?
That if we take an index greater than 'z' index, we should get 'a'.
Let us see :

→ WE WILL USE MODULUS OPERATOR (%)
you all know how modulus operator works
it gives back the remainder.

For example, if 10 % 3 , it will return 1

→ So, what is the no. that we should use for alphabets.
Obviously, it should be 26 (total no. of alphabets)

CODE :

```
alpha = 'abcdefghijklmnopqrstuvwxyz'

i = 25
print ( i % 26)                # will return 25
print ( alpha [i % 26])        # will return alpha[25]

j = 28
print ( alpha [j % 26])

k = 26
print ( alpha [k % 26])        ⇒ 26 % 26 = 0
                               ⇒ alpha[0] ⇒ a
```

OUTPUT :

```
25
z
c
a
```

Now, let us move on to Caesar Cipher.

INTRODUCTION

In this technique, each character is substituted by a certain fixed no. position it's later or before the alphabet.
For example, Alphabet B is replaced by two positions down D. D would become F and so on.

This method is named after popular fictional character Julius Caesar, who used it to communicate with officials.

There is an algorithm used to implement it.

Features of Caesar Cipher Algorithm

The algorithm consists of a few features :

(1) This technique is quite simple to apply encryption.

(2) Each text is replaced by the fixed number of position down or up with the allphabet

(3) It is a simple type of substitute cipher

There is an integer value required to define each letter of the text that has been moved down. The integer value is known as 'shift'.

LET US CODE IN PYTHON NOW :

Let us write a code that shifts the letters of our name one place down (+1).

CODE :

```
alpha = "abcdefghijklmnopqrstuvwxyz"

name = 'gagneet'        # output should be → hbhoffu

# we will take help of alpha variable
```

```
copy = ' '

i = 0
k = 1          # shift

copy = copy + alpha[ ((alpha.index(name[i])) + k) % 26]
print (copy)
# what alpha.index[] does?
# It gives index of particular letter in alpha variable

# What is happening here?
#  name[i] → value of 'name' variable at ith index
# eg. name[0] → g

# now,  alpha.index(name[i])
# it gives the index of letter 'g', for eg., in 'alpha' var
# so, if alpha.index(name[0]) will give 6.

# Now, why +1 of +k ?
# k is the shift we want to make in our program

# We set k=1
# so, (alpha.index(name[0])) + k will return 7
```

$\boxed{\text{as it is 6}}$ therefore

LET US SEE THE OUTPUT :

h

→ So what happened, let us understand digramatically

alpha[((alpha.index(name[i])) + k) % 26]

this returns the value of name variable at i^{th} index

This gives us the index of the value, which we got from name variable, in the alpha variable,

For eg. name[0] = g

alpha.index(name[0]) = 6

So, we have :

alpha[((alpha.index(name[0])) + 1) % 26]

→ alpha[((alpha.index(g)) + 1) % 26]

→ alpha[(6 + 1) % 26]

→ alpha[7 % 26] which is 7

→ It will return → 'h'

COMPLETE CODE :

```
alpha = 'abcdefghijklmnopqrstuvwxyz'

n = 'sky'
copy = ''

i = 0
k = 1

copy = copy + alpha[((alpha.index(n[i])) + k) % 26]
copy = copy + alpha[((alpha.index(n[i+1])) + k) % 26]
copy = copy + alpha[((alpha.index(n[i+2])) + k) % 26]
print (copy)
```

OUTPUT :

```
tlz
```

Date May 17, 2021

Conditional Statements

If ..... Else ..... Elif .....

Decision making is the most important aspect of almost all programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision.

In python, decision making is performed by the following statements :

1) If statement

2) If Else Statement

3) Nested If statement

The if Statement

The if statement executes a group of statements if the given expression is True.

SYNTAX :

```
if expression:
    statement(s)
```

Let's have a very simple example,

classmate
Date
Page

CODE :

```
x = 10
y = 5

if x > y :
    str = 'Hello World'
    print (str)
```

OUTPUT :

NOTE → The indentation of the group of statements is very important.

If they don't have the same indentation level, you will get an error :

```
x = 10
y = 5
if x > y :
str = "Hello World"      # Will show indentation error
    print (str)
```

The if...else Statement

The else statement executes a group of statements if the given expression to if is False.

SYNTAX :

```
if expression :
    statement(s)
else :
    statement(s)
```

Example :

```
age = 15

if age > 18 :
    print ("user is old enough to enter")
else :
    print ("user is NOT old enough.")
```

The if...elif Statement

The elif statement executes a group of statements if its given expression is True and the first expression False!

SYNTAX :

```
if expression :
    statement (s)
elif expression :
    statement(s)
```

Example :

```
x = 5

if x > 10 :
    print ("Yes")
elif x < 10 :
    print ("No")
```

→ We can also add an else statement to the if...elif statement

In this case, the else is executed if both expressions given to if and elif is False.

Let's look at an example;

```
x = 10

if x > 10:
    print ("x is more than 10")
elif x < 10:
    print ("x is less than 10")
else:
    print ("x is 10 because it's neither)
```

* Using Logical Operators

We can use the <u>and</u> and <u>or</u> operators with conditional statements.

- The <u>and</u> operator returns <u>True</u> if both expressions (or Boolean values) are True.

<u>CODE:</u>

```
x = 10
y = 5

if (x == 10) and (y == 5):
    str = "Hello World"
    print (str)
```

<u>OUTPUT:</u>

- The <u>or</u> operator returns <u>True</u> if either of the expressions (or Boolean values) is True.

<u>CODE:</u>

```
x = 10
y = 5
```

```
if (x==10) or (y==15):
# str {
  str = "Hello World!"
  print (str)
```

★ Nested If Statements

An if statement can be placed inside another if statement.

This is called nested if statements.

```
x = 15

if (x >5):
  print ("x is more than 5")
  if (x>10):
     print ("x is more than 10")
  else:
     print ("x is between 5 and 10")
```

Python if...else Shorthand

A shorthand if...else statement is used when you only have one statement to execute.

This can be done using the ternary or conditional operator.

Syntax:

<on_true> if <expression> else <on_false>

- To better demonstrate this, let's convert an if...else statement to its shorthand version.

CODE:
```
x = 10

if x == 10:
    print ("x is 10")
else:
    print ('x is not 10')
```

→ This is the shorthand version:

CODE:
```
x = 10
print ("x is 10") if x == 10 else print ('x is not 10')
```

- Another use case of this is when you want to assign a value to a variable depending on the result of an if...else statement.

In this example, the "x is 10" string will be assigned to the str variable because the given expression is True.

```
x = 10
str = "x is 10" if x == 10 else "x is not 10"
print (str)
```

Let's Practice

Problem 1: Find whether the given number is even or odd.

CODE:
```
x = int ( input ( ) )

if x % 2 = = 0 :
    print ('Even')
else:
    print ('Odd')
```

Problem 2: Find whether the given number ends with 0 or 5 or any other number.

CODE:
```
x = int ( input ('Enter a number'))

if x % 5 = = 0 :
    if x % 10 = = 0 :
        print ('0')
    else:
        print ('5')
else:
    print ('Other')
```

Problem 3: Find the grade of student based on the given marks from 0 to 100

Student Grades

S.no.	Grade	Marks Range
1	A	≥90
2	B	≥80 and ≤90
3	C	≥70 and <80
4	D	≥60 and <70
5	E	~~x~~ <60

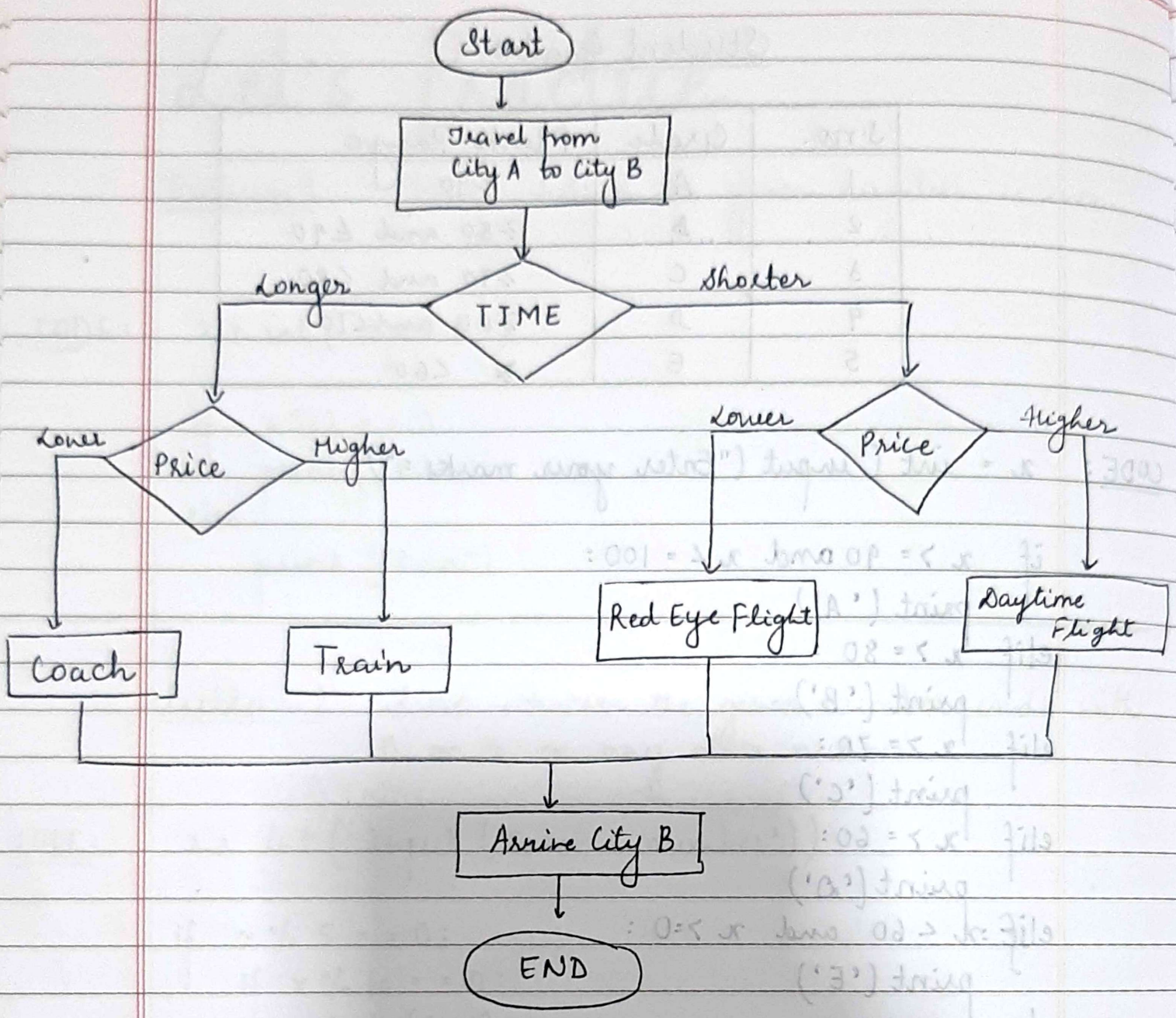
CODE:

```
x = int ( input ("Enter your marks :")

if  x >= 90 and x <= 100 :
    print ('A')
elif  x >= 80 :
    print ('B')
elif  x >= 70 :
    print ('C')
elif  x >= 60 :
    print ('D')
elif x < 60  and  x >=0 :
    print ('E')
else :
    print ('Not Valid')
```

Problem 4: Convert the given flowchart into a Python Code

...contd.



CODE:

```
print ('Travel from City A to City B')
Time = int (input ('Enter Time :'))
Longer = int (input ('Define Longer :'))
if (Time >= Longer) :
    Price = int (input ('Enter price:'))
    Higher = int (input ('Define higher :'))
    if (Price >= Higher) :
        print ('Train')
else :
```

```
        print ('Coach')
else :
    Price = int (input ('Enter a Price :'))
    Higher = int ( Input ('Define Higher :'))
    if (Price >= Higher) :
        print ('Daytime Flight')
    else :
        print ('Red Eye Flight')
print ('Arrive City B')
```

May 18, 2021

Importing Libraries

In python, libraries are known as 'MODULES'

What Is A Module?

Consider a module to be the same as a code library. A file containing a set of functions you want to include in your application.

Use A Module

The 'import' statement is used to import all the functionality of one module into our code.

Here, we must notice that we can use the functionality of any python source file by importing that file as the module into another python source file.

Let us import the math module in our code:

CODE: ~~import math;~~ import math

~~print (pow (10,3))~~ print (math.pow (10,3))

Importing parts of Module : We can choose to import only some parts of a module

Use from keyword to import a part of the module.

Python has many useful built-in modules that we can use to make coding easier.

(1) Math Module

The math module gives us access to mathematical functions.

To use math module, import it first:

Syntax: import math

Now we can start using it.

One of the most useful functions of math module is sqrt() function.

→ The math.sqrt() function returns the square root of the given number.

CODE:
```
import math

x = math.sqrt(9)
print(x)
```
OUTPUT: 3

→ The math.pow(x, y) function returns the x raised to the power y.

CODE:
```
import math
x = math.pow(2,3)
print(x)
```
OUTPUT: 8

→ The math.ceil() method rounds up a number to its nearest integer.

→ The math.floor() methods rounds down a number to its nearest integer.

CODE:

```
import math

x = math.ceil(2.6)
y = math.floor(2.6)

print (x)
print (y)
```

OUTPUT : 3
2

MATH CONSONANTS

The math module also contains math consonants like the pi and e.

CODE:

```
import math
x = math.pi
y = math.e
print (x)
print (y)
```

OUTPUT: 3.1415926535
2.7182818284

TRIGNOMETRIC FUNCTIONS

The math module also contains trignometric functions:

math.cos(x) ; math.sin(x) ; math.tan(x)

(2) Random Module

The random module lets us generate a random number. To use random module, import it first:

SYNTAX: import random

Now, we can start generating a random number.

→ The random.random() method generates a random floating point in the range of 0.0 to 1.0

CODE:
```
import random
x = random.random()
print(x)
```

→ The random.randint(x,y) method generates a random integer in the range of x to y.

CODE:
```
import random
x = random.randint(1,10)
print(x)
```

(3) Calendar Module

Calendar module allows output calendars like the program and provides additional useful functions related to the calendar.
By default, these calendars have monday as the first day of week (the European convention)

Example 1 : Display calendar of given month.

```
import calendar
print(calendar.month(2020,10))
```

Output:

```
      October 2020
Mo Tu We Th Fr Sa Su
              1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

Example 2 : Display calendar of given year.

```
import calendar
print(calendar.calendar(2018))
```

OUTPUT : It will print the complete calendar of 2018.

Importing Modules In Different Ways :

(1)
```
# import module
import math
```

(2)
```
# from module import mod1
from calendar import month
x = month(2020, 10)
print(x)
```

(3)
```
# using alias
import math as m
print(m.sqrt(9))
```

(4) from module import *

SUMMARY OF WEEK 2

- Rules of writing variable name
- Use of comments (#) symbol
- Dynamic Typing in Python
- Multiple Assignment
- Deleting A Variable
- Shorthand Operators
- Importing Modules or Libraries
- Escape Character
- Use of Diff. Types of Quotes ('', "", ''' ''')
- String Methods
  - lower()
  - upper()
  - isupper()
  - isalpha()
  - lstrip()
  - endswith()
  - replace()
  - capitalize()
  - title()
  - istitle()
  - isalnum()
  - rstrip
  - count()
  - swapcase()
  - islower()
  - isdigit()
  - strip()
  - startswith()
  - index()
- Caesar Cipher Algorithm & code
- Conditional Statements
  - If
  - If... else
  - Elif