23-09-2024

Training Day- 6

Data Types: **Are used to save or represent different types of values**.

Single Element/Single Valued: In Python, **single element** or **single-valued** structures or variables refer to data that contains exactly one item or value

int: Whole number: 2000, -80

float: Decimal point numbers: 5.0, 2.35

complex: complex numbers: real + imaginary: 2+3j

bool: True, False

NoneType: None

Multi Element/Multi Valued/ Iterators: len function can work only on iterators

str: single, double or triple quotes

list

tuple

dict

set

frozenset

Variables: Whose value vary in the program. Variables are data

storing elements: Variables are stored in RAM: RAM is a volatile memory. We can store the values in variables

EXAMPLE

```
x=5     #x is a variable of int type
x=2+3j  #x is a variable of complex type
x=True  #x bool type
```

Possible arguments in print:

print(values, variables, expressions, conditions, functions, classes)

……..New Program……..

```
# print(2000)
# print(2.5)
# print("CETPA")
```

……..New Program……..

```
# x="x"
# print(x,"x")
```

How Variables Are Created In Python: By assigning the values

```
x=5     #x int type variable
x="cetpa"   #x str type variable
```

……..New Program……..

```
# x=y     #NameError: name 'y' is not defined
# print(x)
```

……..New Program……..

```
# true=5
# print(true)
```

……..New Program……..

```
# 7=5     #SyntaxError:
```

……..New Program……..

```
# True=5          #SyntaxError: cannot assign to True
# print(True)
```

……..New Program……..

```
# x=true
# print(x)
```

……..New Program……..

```
# name="tiger"
# print("Name:",name)
```

"""

Possible arguments in print:

print(values, variables, expressions, conditions, functions, classes)

"""

……..New Program……..

```
# a,b=3,4
# s="CETPA"
# print(23,True,a,s,a+b,a>b,len(s),type(s))
```

……..New Program……..

```
# s="Welcome to company"
# print(len(s))
```

……..New Program……..

```
# x=2+3j
# print(type(x))
# print(x)
```

……..New Program……..

```
# x="CETPA"
# print(type(x))
# print(x)
```

STRINGS:

Single Line String: Single quote, double quotes or triple quotes

Multi Line String: Only triple quotes are allowed to make the strings

```
# x='Welcome to CETPA'
# print(x)
# x="Welcome to CETPA"
# print(x)
```

```
# x="'Welcome to CETPA"'
# print(x)
# x="""Welcome to CETPA"""
# print(x)
```

How To Take The Data From The User Or From The Screen:

We have a Radymade function**: input ( )**

Syntax:

```
input("Message for user")
var=input("Message for user")
```

……..New Program……..

```
# x=input("Enter Your Name:")
# print(x)
```

……..New Program……..

```
# x=input("What are you doing?")
# print(x)

# #New Program: future concept
# x=input("Enter 5 numbers: ").split()
# print(x)
# print(len(x))
```

input function always returns str type data in our program

```
"""
# x="5"
# print(x)

# #New Program
# x=input("Enter Your Name:")
# print(x,type(x))
# x=input("Enter Any Number:")        #x="1000"
# print(x,type(x))
```

……..New Program……..

Incorrect addition

```
# a=input("Enter First No:")  #a=5
# b=input("Enter Second No:") #b=7
# s=a+b
# print(s)
```

……..New Program……..

Incorrect addition

```
# a=input("Enter First No:")  #a="5"
# b=input("Enter Second No:") #b="7"
# s=a+b       #s="57" # print(s)
```

24-09-2024

Training Day- 7

Unpacking in Python is a way to assign the elements of an iterable (like a list, tuple, or dictionary) to multiple variables in a single operation. Python uses unpacking to simplify and make the code cleaner. Here's a detailed explanation

Python support Unpacking directly

```
"""
# a,b,c,d,e="CETPA"
# print(a)
# print(b)
# print(c)
# print(d)
# print(e)
```

……..New Program…….

```
# a,b,c,d="CAT"       #ValueError
# print(a)
# print(b)
# print(c)
# print(d)
```

Whenever we call a print function then it prints the space separated arguments on the screen ie space is automatically in between the arguments, in parallel after printing all the arguments, a new line in printed ie cursor automatically moves to next line after print statement is executed.
space in print function is printed on the screen automatically
if there are more than 1 arguments, space is made to separate the arguments on the screen.

……..New Program…….

```
# a,b,c,d,e=2,3,4,5,6
# print(a,b,c)
# print(d,e)
```

……..New Program…….

```
# print()
# print()
# print()
```

Escape Characters in Python: These characters leaves behind a special functionality on the screen but in actual these characters are not printed itself on the screen
\t: Tab Character
\n: New Line Character

……..New Program…….

```
# s="CE\tTP\tA"
# print(s)
```

……..New Program…….

```
# s="CE   TP  A"
# print(s)
```

……..New Program…….

```
# s="CE\nt\n\nPA"
# print(s)
```

```
# #New Program
# s="CETPA"
# print(s)
```

```
# #New Program
# s='CETPA'
# print(s)
```

```
# #New Program
# s='"CETPA"'
# print(s)
```

```
# #New Program
# s="'CETPA'"
# print(s)
```

Optional Parameters In Print Statement: 'end' and 'sep'

Default value of end = '\n'
Default of sep = ' ' ie space
We can consider end and sep as variables in python. When we print our arguments then in between the arguments automatically sep is printed and at the end of the arguments end is printed.

……..New Program…….

```
# a,b,c,d,e=1,2,3,4,5
# print(a,b,c)        #asepbsepcend  print(a,sep,b,sep,c,end)
# print(d,e)          #dsepeend    print(d,sep,e,end)
```

……..New Program…….

```
# a,b,c,d,e=1,2,3,4,5
# print(a,b,c,sep="*",end="$")
# print(d,e,end="@")
```

SANDBOX: Development Environment

PRODUCTION ENVIRONMENT: Live Environment publish

```
# #New Program
# s="ce't'p'a"
# print(s)
```

……..New Program…….

```
# a,b,c,d,e=1,2,3,4,5
# print(a,end=" ")       #end space
# print(b,end=" ")
# print(c,end=" ")
# print(d,end=" ")
# print(e,end="")     #end empty string
```

……..New Program…….

```
# a,b,c,d,e=1,2,3,4,5
# print(a,end="")       #end empty string
# print(b,end="")
# print(c,end="")
# print(d,end="")
# print(e,end="")     #end empty string

# #New Program
# a,b,c,d,e=1,2,3,4,5
# print(a,b,c,d,e,sep="")
```

……..New Program…….

```
# print("\tCETPA")
# print("C\tCETPA")
# print("CE\tCETPA")
# print("CET\tCETPA")
# print("CETP\tCETPA")
# print("CETPA\tCETPA")
```

Variable: Data Storing Element whose value varies in the program.
How variables are created in Python: By assigning the value

```
"""
# a=b     #NameError: name 'b' is not defined
# print(a)
```

……..New Program…….

```
# a=5
# print(a)
```

Predefined Names: Keywords

User Defined Names: Identifiers

Examples of Identifiers: Variable Names, Function Names, Class Names

Rules To Define Identifiers In Python: Possible valid identifiers:

1. All English Alphabets Are Allowed Ie Upper Case And Lower Case
2. Numbers Are Allowed From 0 To 9 But Identifier Name Should Not Start With Numbers. Numbers Can Be Used In Between Or End Of The Identifiers.
3. Special Symbol: Only Underscore Ie _ Is Allowed
4. Can't Have Any Special Symbol Other Than Underscore
5. Can't Have Any Keyword Names.

```
"""
# a b=5         #Not allowed space
# print(a b)
# cetpa=7       #Allowed
# print(cetpa)
# a*b=9         #Not allowed *
# print(a*b)
# true=5        #Allowed
# print(true)
# True=5        #Keywords not allowed
# print(True)
# _a=5          #Allowed
# print(_a)
# __a__="CETPA" #Allowed
# print(__a__)
# 5a="CETPA"      #Not allowed, can't start with numbers
# print(5a)
# a5=100      #Allowed
# print(a5)

# #New Program
# _5a=6
# print(_5a)

"""
Assignment:
8.        If i = 3, j = 2 what is the result of following expressions?
a.        i + 5 >= j – 6
b.        j * 10 < i ** 2
c.        i < j + 5 > j ** 4


"""
# #New Program
# i = 3
# j = 2
# print(i + 5 >= j - 6)   #8>-4
# print(j * 10 < i ** 2)      #20 <9
# print(i < j + 5 > j ** 4)   #    3<7 >16

# #New Program
# # 7.   If the radius of a circle is 3 meters, find the area of the circle.
# r=3
# a=3.14*r*r
# print(a)
```

25-09-2024

Training Day- 8

CONDITIONAL STATEMENTS: To Execute Set Of Statements On The Basis Of Some Conditions. Conditional Statements In Python Allow You To Control The Flow Of Your Program Based On Conditions. They Are Used To Execute Specific Code Blocks When Certain Conditions Are Met. Python Supports The Following Conditional Statements

. `if` Statement

- Executes a block of code if the condition evaluates to `True`.
- Example:

```
python
Copy code
if x > 0:
    print("x is positive")
```

2. `if-else` Statement

- Provides an alternative block of code to execute when the condition evaluates to `False`.
- Example:

```
python
Copy code
if x > 0:
    print("x is positive")
else:
    print("x is non-positive")
```

3. `if-elif-else` Statement

- Allows checking multiple conditions in sequence.
- The first condition that evaluates to `True` is executed, and the rest are skipped.
- Example:

```
python
Copy code
if x > 0:
    print("x is positive")
elif x == 0:
    print("x is zero")
else:
    print("x is negative")
```

4. Nested `if` Statements

- Places one `if` statement inside another to handle more complex scenarios.

- Example:

```
python
Copy code
if x > 0:
    if x % 2 == 0:
        print("x is a positive even number")
```

Conditional Statements: ***3 Keywords:*** These keywords plays the role of a heading: Heading in python is a statement which is having some sub-statements inside it or a block of code inside it.

Syntax To Create A Heading In Python:

HEADING_NAME:

All the statements inside a heading will be indented statements.

Indentation: Statements at a fixed gap (spacebars) w.r.to heading

If condition will be true then only statements will be executed.

if is a condition

elif is a condition

else executes if conditions above else are false

"""Statements inside a heading ie the block of code inside heading in python is called a suite.

There should be al least one statement inside a heading.

"""

```
# #New Program
# x=5
# if(x<=5):
#     print("CETPA")
#     print("Welcome")
# print("ABCD")
```

Single Liner Headings: Then we can mention the statement directly next to heading in same line

……New Program……..

```
# x=5
# if(x==5):print("CETPA")
```

```
# print("Welcome")
```
……New Program……..
```
# x=5
# if(x==5):pass
# print("Welcome")
```
……New Program……..
```
# x=5
# if(x==5):
#   pass
#   print("CETPA")
# print("Welcome")
```
……New Program……..
```
# x=5
# if(x==5):
#   pass
# print("CETPA")
# print("Welcome")
```
……New Program……..
```
#   print("CETPA")    #IndentationError: unexpected indent
#   print("Hello")
```
……New Program……..
```
# id=int(input("Enter the ID:"))
# if(id==1000):
#   print("Welcome to the System")
# else:
#   print("You are not allowed the entry")
```
……New Program……..
```
# id=input("Enter the ID:")
# if(id=="1000"):
#   print("Welcome to the System")
# else:
#   print("You are not allowed the entry")
```

MULTIPLE CONDITIONS: if, elif, else

elif and else can't work without if

elif: else if

"""

```
# day=input("Enter the day:")
# if(day=="Sunday"):
#   print("Take Rest")
# elif(day=="Saturday"):
#   print("Go to Movie")
# elif(day=="Friday"):
#   print("Go for Shopping")
# else:
#   print("Go for CETPA Class")
```

If we want to check multiple conditions in single heading:

we can use logical operators 'and' and 'or'

……New Program……..

Better way

```
# day=input("Enter the day:")
# if(day=="Sunday"):
#   print("Take Rest")
# elif(day=="Saturday"):
#   print("Go to Movie")
# elif(day=="Friday"):
#   print("Go for Shopping")
# elif(day=="Monday" or day=="Tuesday" or day=="Wednesday" or day=="Thursday"):
#   print("Go for CETPA Class")
# else:
#   print("Incorrect input")
```

"""

NESTED CONDITIONS: Conditions inside conditions:

heading inside heading.

If there is a heading inside heading, then statements of the inner

heading will be indented with respect to inner heading

"""

……New Program……..

```
# a=5
# b=9
# if(a==5):
#   print("CETPA")
#   print("ABCD")
#   if(b==7):
#     print("Welcome")
#     print("PQRS")
#   print("UVWR")
# else:
#   if(a==6):
#     print("1234")
```

……New Program……..

```
# a=5
# b=7
# if(a==5):
#   print("CETPA")
#   print("ABCD")
#   if(b==7):
#     print("Welcome")
#     print("PQRS")
#   print("UVWR")
# else:
#   if(a==6):
#     print("1234")
```

……New Program……..

```
# a=6
# b=7
# if(a==5):
#   print("CETPA")
```

if, elif we get True inside condition then block will execute else won't

"""

……New Program……..

```
# a,b=5,7
# print(a==b)
# print(a<b)
# print(a!=b)
```

……New Program……..

```
# a,b=5,5
# if(a==b):
#   print("CETPA")
```

False Values in Python: In Python, a **false value** refers to any value that evaluates to `False` in a boolean context, such as when used in conditional statements. These are called **falsy values**. Here's a complete explanation:

Falsy Values in Python

The following are the values that are treated as `False`:

0

False

None

All empty values

Rest all are True values

……New Program……..

```
# if(55):
#   print("CETPA")
```

……New Program……..

```
# if(""):
#   print("CETPA")
```

26-09-2024 # Training Day- 8

SWAPPING IS VARIABLE

 # #Using 3rd Variable

```
# a=5
# b=7
# temp=a      #3rd Varaible is temp, temp=5, b=7, a=5
# a=b         #a=7, temp=5
# b=temp
# print(a,b)


# #New Program
# a,b=5,7
# a,b=b,a
# print(a,b)
```

PASS :- pass is an instruction in python, which does nothing

There are places in python, where we must write some block of code, like inside a heading, there for the time being we can write pass

```
"""
# #New Program
# pass
# pass
# pass
# pass
# pass


# #New Program
# x=6
# if(x==5):
#     print("CETPA")


# #New Program
```

```
# x=6
# if(x==5):
#     print("CETPA")
# else:
#     pass
```

PROGRAM/APPLICATION/SOFTWARE/COMPUTER: To take the date from the user, process the data and generate the outputs.

```
"""
# print()
# print()
```

Take two inputs from the user and check their data types.

```
"""
# #New Program
# x=input("Enter A Number:")
# print(type(x))
# x=input("Enter A String:")
# print(type(x))

# Example: Find the bigger of 2 numbers without using logical operators
# """
# #New Program
# no1=int(input("Enter First No:"))       #no1=5
# no2=int(input("Enter Second No:"))      #no1=7
# if(no1>no2):
#     print(no1, "is bigger")
# else:
#     print(no2,"is bigger")



"""
Example: Find the biggest of 3 numbers without using logical operators
or using nested conditions
```

```
"""
# #New Program
# no1=int(input("Enter First No:"))       #no1=5
# no2=int(input("Enter Second No:"))      #no1=7
# no3=int(input("Enter Third No:"))       #no1=9
# if(no1>no2):        #no2 is not the biggest no
#     if (no1 > no3):
#         print(no1, "is the biggest no")
#     else:
#         print(no3, "is the biggest no")
# else:               #no1 is not the biggest no
#     if (no2 > no3):
#         print(no2, "is the biggest no")
#     else:
#         print(no3, "is the biggest no")

"""
```

Find the biggest of 3 numbers using logical operators or without using nested conditions.

```
"""
# #New Program
# no1=int(input("Enter First No:"))       #no1=7
# no2=int(input("Enter Second No:"))      #no2=5
# no3=int(input("Enter Third No:"))       #no3=9
# if(no1>no2 and no1>no3):
#     print(no1,"is the biggest no")
# elif(no2>no3):                  #no1 is not the biggest no
#     print(no2,"is the biggest no")
# else:
#     print(no3, "is the biggest no")

"""
```

Example: Find the bigger of 2 numbers and also check whether they are equal or not

"""

```
# #New Program
# no1=int(input("Enter First No:"))       #no1=5
# no2=int(input("Enter Second No:"))      #no1=7
# if(no1>no2):
#     print(no1, "is bigger")
# elif(no2>no1):
#     print(no2,"is bigger")
# else:
#     print("Both the numbers are equal")
```

Makes The Program Scalable.

To make the project scalable, we divide the project into multiple modules and layers.

Layers are further divided into classes. Classes are made up of functions and variables.

ERP: Enterprise Resource Planning

The functions which are outside class are called functions only,

how to call them: function_name(arguments)

The functions which are inside class are called methods or functions, how to call method or a

function made inside class: obj_name.method_name()

How To Create Object Of Any Class In Python:

Standard syntax:

obj_name=class_name()

or

obj_name=class_name(arguments)

Above syntax will create a default valued object of the class

s=str() #variable or object of string class

Default value of string class is emtpy string

n=int() #variable or object of int class

Default value of int class is 0

…….New Program…….

```
# s=str()     #
# print(s)
# print(type(s)
```

…….New Program…….

```
# f=float()     #
# print(f)
# print(type(f))

# #New Program
# s="Cetpa"       #Object of string class
# r=s.upper()
# print(r)

# #New Program
# s="cetpa infotech**"
# r=s.title()
# print(r)
```

Split Methods splits a string and generate a list based on criteria given

Default criteria is space

```
"""
# #New Program
# s="cet*pa info*tech"
# r=s.split("*")
# print(r)

# #New Program
# name="Vikas Kumar Kalra"
# name=name.split()
# print(name)
```

27-09-2024

Training Day- 9

IDENTIFIERS VS VALUES:

IDENTIFIERS: User defined names: Set of rules:

Using Combination Of Following Characters:
1. English alphabets: A to Z, a to z
2. Numbers or : 0 to 9
3. underscore, : _

We Create Identifiers
Special symbols other than underscore not allowed, should not start with numbers to create identifiers. keywords can't be identifiers
Identifiers are created directly without using quotes or any special symbol
Python is a case sensitive langauge

EXAMPLES OF IDENTIFIERS: Variable name, function name or class name
Values: Different type of values: Data Types

…….New Program…….
```
# print(100)
# print(2.35)
# print(100,200,300)
```
…….New Program…….
```
# x=100
# temp=200
# flag123=500
# cetpa_2345=900
```
…….New Program…….
```
# a*b=200        #SyntaxError
# a b=500 #Error
# 1ab=82  #Identifier names, can't start with numbers
```
…….New Program…….
```
# a=100
# A=200
# print(a,A)
```
…….New Program…….
```
# Ram=5
# print(ram)      #NameError:
```

IDENTIFIER: Collection of particular characters:
User defined names 1. Variable 2. Functions 3. Classes

KEYWORDS: Predefined names

if, elif, for, def.....
Values: Different types of values: Data types:
Single element:
int 23, -92
float 2.35, 4.0
complex 2+3j
bool True, False
NoneType None

MULTI ELEMENT: Iterators In Python, **multi-element** refers to any data structure or operation that involves multiple elements. This typically applies to collections like lists, tuples, sets, dictionaries, or other iterables where multiple elements can be stored or processed together
str : Collection of characters: using single, double, or triple quotes
list
tuple
dict
set
frozenset
"""
#New Program
a=b #NameError:
print(a)

#New Program
a=5
print(a)
print(type(a))

str: Collection of characters:

"""

…….New Program…….:: Single Line String

a='Welcome to CETPA'
print(a)
a="Welcome to CETPA"
print(a)
a='''Welcome to CETPA'''
print(a)
a="""Welcome to CETPA"""
print(a)

…….New Program…….: Multi Line String

a='''Welcome to CETPA.
Thanks for joining CETPA'''
print(a)
a="""Welcome to CETPA.

Thanks for joining CETPA"""
print(a)
…….New Program…….:
# a="b" #"b" is a value of string type
# print(a)
print(Comma separated arguments)
print(values,variable,expressions,conditions,functions,classes)

"""
a,b=5,7
s="CETPA"
cetpa="cetpa"
print(99,True,None,a,s,a+b,a>b,len(s),type(s))

"""
Escape Characters:
\n : New Line
\t : Tab Character

If we want to print quotes in a string:
double quotes print: put string in single quotes
single quotes print: put string in double quotes
or
put the quotes in front of \
\' : Single quotes
\" : Double quotes
"""
…….New Program…….:
print("A\t\t\tBCD")
print("A\n\n\nBCD")
…….New Program…….:
x="CETPA"
print(x)
…….New Program…….:
print("he said, \"Hello!\"")
"""
String Is A Collection Of Characters:
'abc----zAB----Z012...9@#$'

"""

…….New Program…….:
'cetpa'

```
# print("'cetpa'")
```

…….New Program…….:

```
"cetpa"
# print("'cetpa'")
```

…….New Program…….:

```
"cetpa"
# print("'ce"tpa'")
```

…….New Program…….:

```
"cetpa"
# print('\'ce\'tpa\'')
```

…….New Program…….:

```
"cetpa"
# print('\'ce\'t"pa\'')

"""
```

Python Supports Unicode Formats: ie in string, you can write world's any language : special symbol, currency symbol, alphabets...

…….New Program…….:

```
# print("का")
```

Optional Arguments: end, and sep variables: str type

```
end="\n"
sep=" "
```

Print Function: sep variable is automatically printed in between the arguments and end variable is automatically printed at the end of the arguments.

```
"""
# #New Program
# a,b,c,d,e=1,2,3,4,5
# print(a,b,c)       #asepbsepcend
# print(d,e)         #dsepeend
# print(a,b,c,sep="*",end="#$")    ##asepbsepcend
```

…….New Program…….:

```
# a,b,c,d,e=1,2,3,4,5
# print(a,b,c,sep="*",end="#")
# print(d,e,sep="$")
```

…….New Program…….:

```
# print(sep="*",end="#")
# print()
# print()
```

How to use comments:

```
# use
```

Can make strings in the program
Multiple lines to use # : Shortcut key: cntrl + /
"""
#New Program
print("CETPA")
print("Hello")

LOGICAL OPERATORS: Logical operators in Python are used to perform logical operations on Boolean values, which are either True or False. They are essential for creating conditional statements and expressions, and for implementing complex decision-making processes. Following the all are logical operator
and #English wala and, hindi wala aur
or #Englsih wala or ya hindi wala ya
not #Wahi nahi
and: if both inputs are True then output is True else False
or: if at least one of the inputs is True then output is True else False
"""
……..New Program……..
a,b=5,7
print(a==5 and b==7)
print(a>=5 and b<7)
print(a>=5 or b<7)
……..New Program……..
a,b=5,7
print(a==b)
print(a<b)
……..New Program……..
id=input("Enter the id:")
pwd=input("Enter the pwd:")
if(id=="007" and pwd=="bond"):
print("Correct id and pwd")
else:
print("Incorrect id or pwd")

"""
Packing and unpacking concept
"""
#New Program
a,b=5,7
print(a,b)

#New Program
a,b="AB" #Python supports Unpacking directly
print(a,b)

#New Program
a,b,c,d,e="CETPA" #Python supports Unpacking directly
print(a,b,c,d,e)

```
# #New Program
# a,b,c,d="CETPA"        #Error
# print(a,b,c,d)

# #New Program
# a,b,c,d,e,f="CETPA"        #Error
# print(a,b,c,d,e)

# #New Program
# a="C","E","T","P","A"
# print(a,type(a))

# #New Program
# a=False,False
# print(a)
# print(type(a))

"""
```

Nested Conditions:

Conditions inside conditions

```
"""
# #New Program
# a,b,c=1,2,3
# if(a>=1):
#     print("Inside First If")
#     if(b<=2):
#         print("Inside Second If")
#     elif(b==3):
#         print("Inside First elif")
#     else:
#         print("ABCD")
# elif(a==0):
#     print("BCDE")
#     if(b==1):
#         print("CDEF")

# #New Program
# a=5
# if(a==5):
#     print("CETPA")
# elif(a==5):        #else if
#     print("Hello")
# else:
#     print("ABCD")
```

Comparison of 3 numbers

```
# #Find the bigger between 2 numbers
```

```
# no1=input("Enter First No:")
# no2=input("Enter Second No:")
# if(no1>no2):
#     print(no1,"is the bigger no")
# else:
#     print(no2,"is the bigger no")

# #Find the bigger between 2 numbers or check if they are equal
# no1=input("Enter First No:")
# no2=input("Enter Second No:")
# if(no1>no2):
#     print(no1,"is the bigger no")
# elif(no1<no2):
#     print(no2,"is the bigger no")
# else:
#     print("Both numbers are equal")


# #New Program: Find the biggest of 3 numbers: Using nested conditions
# no1=int(input("Enter First No:"))
# no2=int(input("Enter Second No:"))
# no3=int(input("Enter Third No:"))
# if(no1>no2):    #If no2 is not the biggest, we will go inside if
#     if(no1>no3):
#         print(no1,"is the biggest no")
#     else:
#         print(no3,"is the biggest no")
# else:       #If we reach at else, it means no1 is not the biggest no
#     if (no2 > no3):
#         print(no2, "is the biggest no")
#     else:
#         print(no3, "is the biggest no")

# #New Program
# print("1" > "5")
# print("1" < "5")
# print("11" > "5")
# print(11 > 5)
# print("ABC" > "AAC")

"""

"""
```