



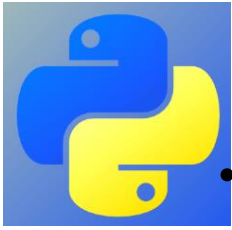
# PYTHON PROGRAMMING



# Fundamentals of Python Programming

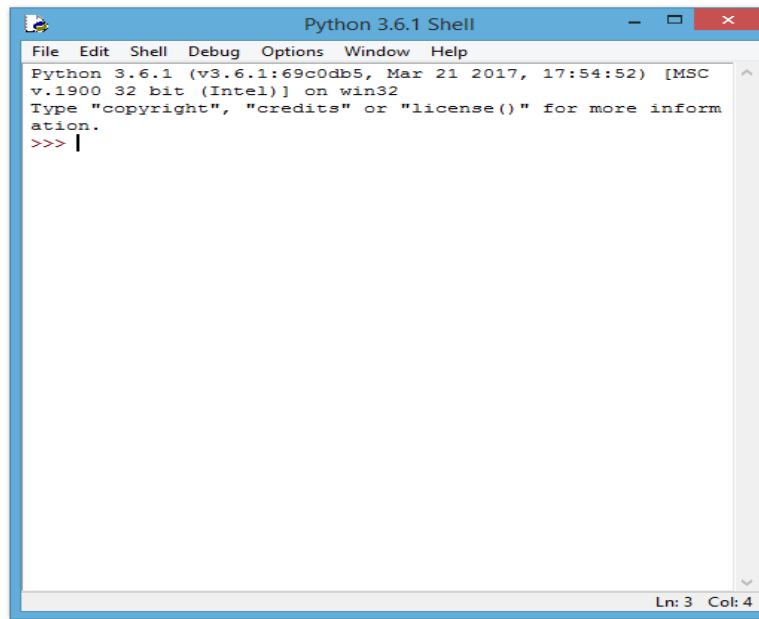
## Salient Features

- Interactive, interpreted, and object-oriented programming language.
- Platform independent.
- Simple syntax, Free and Open source.
- Portable, extensible and expandable.
- Large standard libraries to solve a task.
- Developed by Guido Van Rossum in 1991 at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Name was inspired by: Monty Python's Flying Circus.
- Allows to distinguish input, output, and error messages by different colour codes.

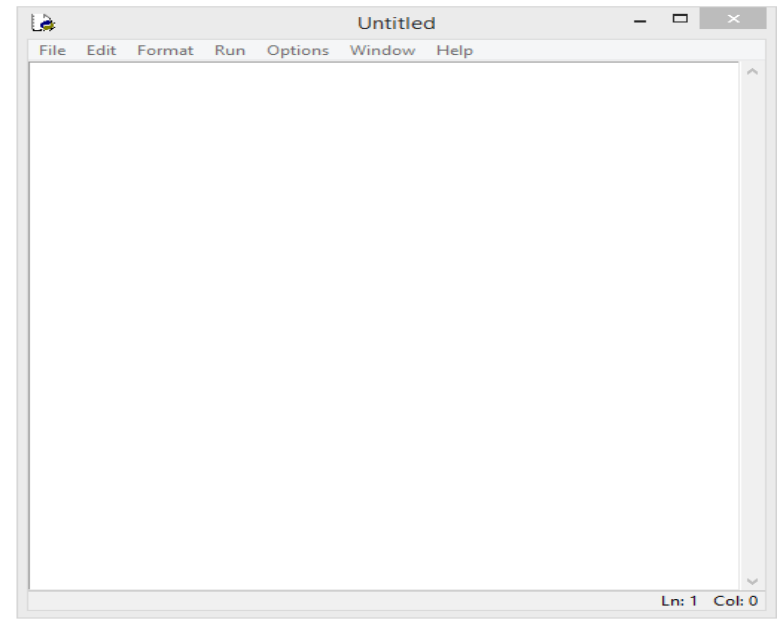


# PYTHON PROGRAMMING ENVIRONMENT

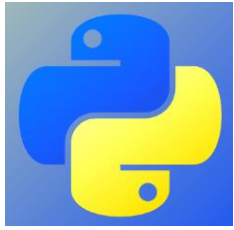
- Available on a wide variety of platforms including Windows, Linux and Mac OS X.
- Official Website: [python.org](https://python.org)
- IDLE stands for Integrated Development and Learning Environment. Python IDLE comprises of Python Shell and Python Editor.



**Python Shell**



**Python Editor**



# Python Keywords

- Reserved words that are already defined by the Python for specific uses.

```
In [ ]: import keyword  
        print(keyword.kwlist)
```

<b>False</b>	<code>class</code>	<code>finally</code>	<code>is</code>	<code>return</code>
<b>None</b>	<code>continue</code>	<code>for</code>	<code>lambda</code>	<code>try</code>
<b>True</b>	<code>def</code>	<code>from</code>	<code>nonlocal</code>	<code>while</code>
<code>and</code>	<code>del</code>	<code>global</code>	<code>not</code>	<code>with</code>
<code>as</code>	<code>elif</code>	<code>if</code>	<code>or</code>	<code>yield</code>
<code>assert</code>	<code>else</code>	<code>import</code>	<code>pass</code>	
<code>break</code>	<code>except</code>	<code>in</code>	<code>raise</code>	



# Sample Python Command Line Expressions

```
>>> 99+10
```

```
109
```

```
>>> 4-6*6
```

```
-32
```

```
>>> "99"+"98"
```

```
'9998'
```

+ with strings concatenates them

```
>>> "70"*3
```

```
'707070'
```

\* with strings and number(s)

```
>>> 2*"Hello"*3
```

```
'HelloHelloHelloHelloHelloHello'
```

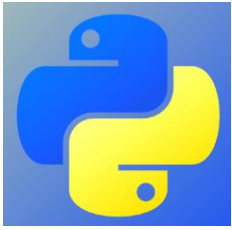
.

## Display on screen

```
In [2]: print('hello world')
```

```
hello world
```

Since, Python is a case-sensitive language so print and PRINT are different.

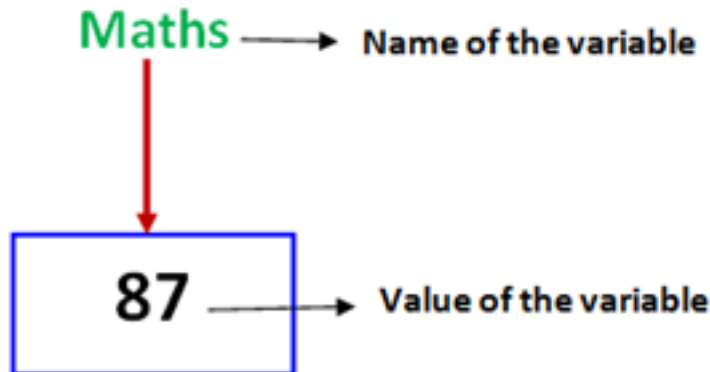


# Names (Variables) and Assignment Statements

- Variables provide a means to name values so that they can be used and manipulated later.
- Assignment Statement: Statement that assigns value to a variable.

```
In [ ]: Maths = 87  
        print(Maths)  
        87
```

Python associates the **name** (variable) **Maths** with value **87** i.e. the name (variable) **Maths** is assigned the value **87**, or that the name (variable) **Maths** refers to value **87**. Values are also called **objects**.





## Multiple Assignments

- Used to enhance the readability of the program.

```
>>> msg, day, time='Meeting','Mon','9'
```

```
>>> print (msg,day,time)
```

```
Meeting Mon 9
```

```
totalMarks = count = 0
```

## Data Types and Associating them with variables

```
Number=10      #Integer
```

```
Price=240.50   #Float
```

```
City="Mumbai"  #String
```

```
Number-=5      #Integer Manipulation
```

```
Price=Price*2  #Integer Manipulation
```

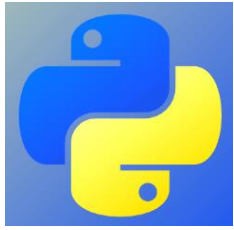
```
City+="Delhi"  #String Manipulation
```

```
print (Number)
```

```
print (Price)
```

```
print (City)
```

```
5
481.0
MumbaiDelhi
```



# Operators in Python

## Arithmetic Operators

```
print("18 + 5 =", 18 + 5)  #Addition
print("18 - 5 =", 18 - 5)  #Subtraction
print("18 * 5 =", 18 * 5)  #Multiplication
print("18 / 5 =", 18 / 5)  #Division
print("27 / 5 =", 27 / 5)  #Integer Division
print("27 // 5 =", 27 // 5) #Modulus
print("27      5 =", 27      5) #Exponentiation
print("2 ** 3 =", 2 ** 3)   #Exponentiation
print("-2 ** 3 =", -2 ** 3)
```

18 + 5 = 23

18 - 5 = 13

18 \* 5 = 90

27 / 5 = 5.4

27 // 5 = 5

27 5 = 2

2 \*\* 3 = 8

-2 \*\* 3 = -8

## Precedence of Arithmetic Operators

() (parentheses)	↓ decreasing order
** (exponentiation)	
- (negation)	
/ (division) // (integer division) * (multiplication) % (modulus)	
+ (addition) - (subtraction)	





## Relational Operators

- Used for comparing two expressions and yield True or False.
- The arithmetic operators have higher precedence than the relational operators.

```
print("23 < 25 :", 23 < 25)
```

*#less than*

```
print("23 > 25 :", 23 > 25)
```

*#greater than*

```
print("23 <= 23 :", 23 <= 23)
```

*#less than or equal to*

```
print("23 - 2.5 >= 5 * 4 :", 23 - 2.5 >= 5 * 4)
```

*#greater than or equal to*

```
print("23 == 25 :", 23 == 25)
```

*#equal to*

```
print("23 != 25 :", 23 != 25)
```

*#not equal to*

- When the relational operators are applied to strings, strings are compared left to right, character by character, based on their ASCII codes, also called ASCII values.

```
print("'hello' < 'Hello' :", 'hello' < 'Hello')
```

```
print("'hi' > 'hello' :", 'hi' > 'hello')
```

**'hello' < 'Hello' : False**

**'hi' > 'hello' : True**

PLEASE NOTE **<>** is  
discontinued after Python 2.7



## Logical Operators

- The logical operators not, and, and or are applied to logical operands True and False, also called Boolean values, and yield either True or False.
- As compared to relational and arithmetic operators, logical operators have the least precedence level.

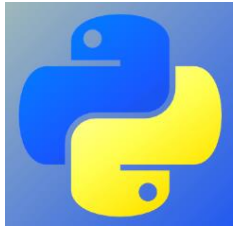
```
print("not True < 25      :", not True)           #not operator
print("10 < 25 and 5 > 6   :", 10 < 25 and 5 > 6)   #and operator
print("10 < 25 or 5 > 6    :", 10 < 25 or 5 > 6)   #or operator
```

## Identity Operators

is	Returns true if LH Operand is found in RH Sequence
is not	Returns true if LH Operand is found in RH Sequence

```
>>> N=10
>>> V=10
>>> print(id(N), id(V))
1916953088 1916953088
>>> N is V
True
```

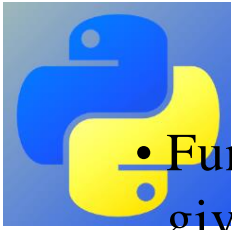
```
>>> N="Raj"
>>> print(id(N), id(V))
2712062284832 1916953088
>>> N is V
False
```



# Precedence of Operators

1	**
2	+ - (Unary Operators)
3	* / % //
4	+ - (Binary Operators)
5	<= < > >=

6	== !=
7	= %= /= //= -= += *= **=
8	is is not
9	in not in
10	(i) not (ii) and (iii) or



# Functions

- Functions provide a systematic way of problem solving by dividing the given problem into several sub-problems, finding their individual solutions, and integrating the solutions of individual problems to solve the original problem.
- This approach to problem solving is called stepwise refinement method or modular approach.

## Built-in Functions

- Predefined functions that are already available in Python.

## Type Conversion: int, float, str functions

```
>>>str(123)
```

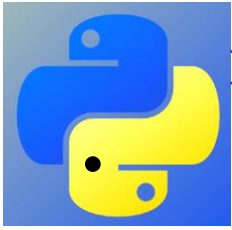
```
'123'
```

```
>>>int('234')
```

```
234
```

```
>>>int(234.8)
```

```
234
```



# input function

Enables us to accept an input string from the user without evaluating its value.

- The function input continues to read input text from the user until it encounters a newline.

```
>>>name = input('Enter your name: ')\n      print('Welcome', name)
```

Enter your name: Yogesh

Welcome Yogesh

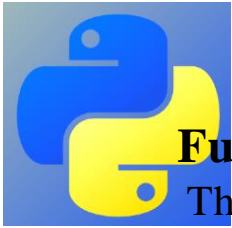
Another  
Example: To find  
the profit earned  
on an item

```
costPrice = int(input('Enter cost price: '))\nprofit = int(input('Enter profit: '))\nsellingPrice = costPrice + profit\nprint('Selling Price: ', sellingPrice)
```

Enter cost price: 50

Enter profit: 12

Selling Price: 62



# User-Defined Functions

## Function Definition and Call

The **syntax** for a function definition is as follows:

```
def function_name ( comma_separated_list_of_parameters):  
    statements
```

Note: Statements below **def** begin with four spaces. This is called **indentation**. It is a requirement of Python that the code following a colon must be indented.

```
def triangle():
```

```
    '''
```

```
    Objective: To print a right angled triangle. Input
```

```
    Parameter: None
```

```
    Return Value: None
```

```
    '''
```

```
    '''
```

```
    Approach: To use a print statement for each line of output
```

```
    '''
```

```
    print('*')
```

```
    print('* *')
```

```
    print('* * *')
```

```
    print('* * * *')
```

## Invoking the function

```
>>>triangle()
```

```
*
```

```
* *
```

```
* * *
```

```
* * * *
```



## Computing Area of the Rectangle

```
def areaRectangle(length, breadth):
```

```
'''
```

*Objective: To compute the area of rectangle Input*

*Parameters: length, breadth      numeric value Return*

*Value: area - numeric value*

```
'''
```

```
    area = length * breadth
```

```
    return area
```

```
>>>areaRectangle(7,5)
```

```
35
```

```
>>>help(areaRectangle)           #help on any function
```

```
Help on function areaRectangle in module_main_:
```

```
areaRectangle(length, breadth)
```

Objective: To compute the area of rectangle

Input Parameters: length, breadth      numeric value

Return Value: area - numeric value



# Conditional Statements

Syntax:

```
if (Condition):  
    Statement
```

```
Age=int(input("Age:"))  
if (Age>=5):  
    print("In School")
```

```
Age: 6  
In School
```

---

```
Q=input("Y/N:")  
if (Q=="Y"):  
    print("YES")
```

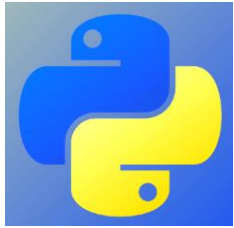
```
Y/N: Y  
YES
```

---

```
N=int(input("Number:"))  
if (N):  
    print("Non Zero")
```

```
Number: 45  
Non Zero
```





## Conditional Statements

```
Age=int(input("Age:"))  
if (Age>=5):  
    print("In School")  
else:  
    print("Stay at Home")
```

```
Q=input("Python - Snake?")  
if (Q=="Y"):  
    print("Right")  
else:  
    print("Good - It is a language")
```

Syntax:

```
if (Condition):  
    Statement1  
else:  
    Statement1
```

Age:4

Stay at Home

Python - Snake?N

Good - It is a language



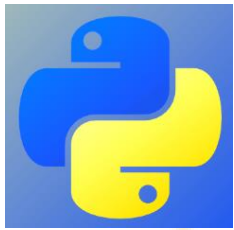
## Conditional Statements

```
Age=int(input("Age:"))
if (Age>=18):
    print("In College")
elif (Age>=5):
    print("In School")
else:
    print("Stay at Home")
```

Syntax:

```
if (Condition):
    Statement1
elif (Condition):
    Statement2
:
else:
    StatementN
```

Age:23  
In College



## Loops - while

1 `A,B=0,1`  
`while (A<=13):`  
    `print(A)`  
    `A,B=A+B,A`

2 `A,B=0,1`  
`while (A<=13):`  
    `print(A)`  
    `A=A+B`  
    `B=A`

NOT SAME

### Syntax

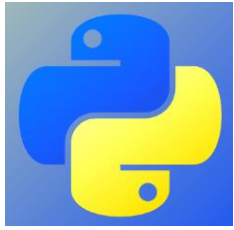
```
while <condition>:  
    <statement>  
    <statement>  
else:  
    <statement>
```

1

0
1
1
2
3
5
8
13

2

0
1
2
4
8



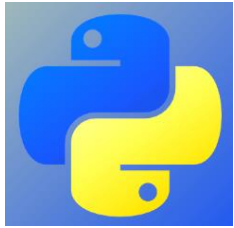
## Loops - while

```
Num=10
while (Num<=17):
    print(Num)
    if (Num%7==0):
        break
    Num=Num+1
else:
    print("Reached Last")
print("Finally")
```

#This else is for break and not for if statement

### Syntax

```
while <condition>:
    <statement>
    <statement>
else:
    <statement>
```



## Loops - for

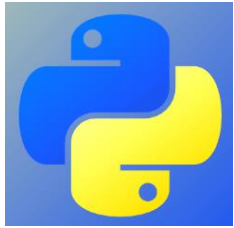
```
for Num in range(5,9):  
    print(Num)
```

```
for Num in range(5,11,2):  
    print(Num)
```

```
for Name in ['Ramesh', 'Suraj', 'Priya']:  
    print(Name)
```

### Syntax

```
for <Variable> in <List/Range>:  
    <statement>  
    <statement>  
else:  
    <statement>
```



## Loops - for

```
for Name in ['Ramesh', 'Suraj', 'Priya']:  
    print(Name)  
    if Name[0]=='Y':  
        break  
else:  
    print('All done!')  
print('Thanks!')
```

```
Ramesh  
Suraj  
Priya  
all done  
thanks  
>>>
```



# Strings, Lists, Tuples and Dictionary



# Strings

- A string is a sequence of characters.
- A string may be specified by placing the member characters of the sequence within quotes (single, double or triple).
- Triple quotes are typically used for strings that span multiple lines.

```
>>> message = 'Hello Gita'
```

## ❖Computing Length using len function

```
>>> print(len(message))
```

```
10
```

## Indexing

- Individual characters within a string are accessed using a technique known as indexing.

Non-negative indices	0	1	2	3	4	5	6	7	8	9
	H	e	l	l	o		G	i	t	a
Negative indices	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> index = len(message) - 1
```

```
>>> print(message[0], message[6], message[index], message[-1])
```

```
H G a a
```

```
>>> print(message[15])
```

```
IndexError Traceback (most recent call last)
```

```
<ipython-input-4-801df50d8d1> in <module>()
```

```
----> 1 print(message[15])
```





## Slicing

- In order to extract the substring comprising the character sequence having indices from start to end-1, we specify the range in the form start:end.
- Python also allows us to extract a subsequence of the form *start:end:inc.*

```
>>> message = 'Hello Sita'
>>> print(message[0:5], message[-10:-5])
Hello Hello
>>> print(message[0:len(message):2])
>>> print(message[:])
HloSt
Hello Sita
```

## Membership Operator in

- Python also allows us to check for membership of the individual characters or substrings in strings using in operator.

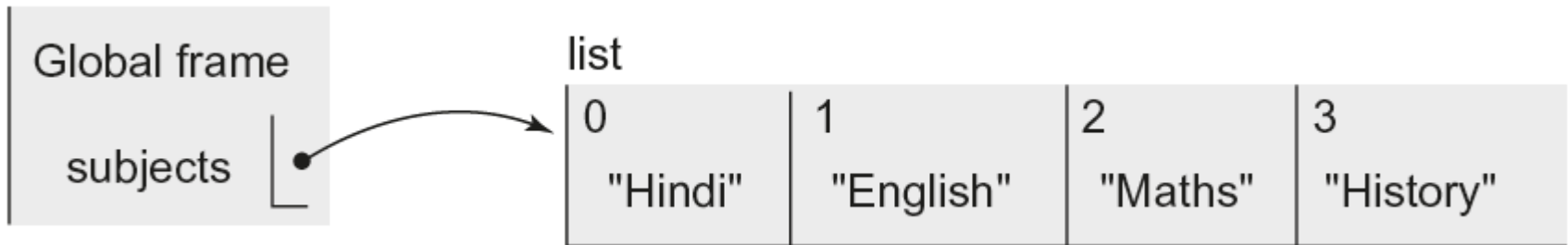
```
>>> 'h' in 'hello'
True
>>> 'ell' in 'hello'
True
>>> 'h' in 'Hello'
False
```



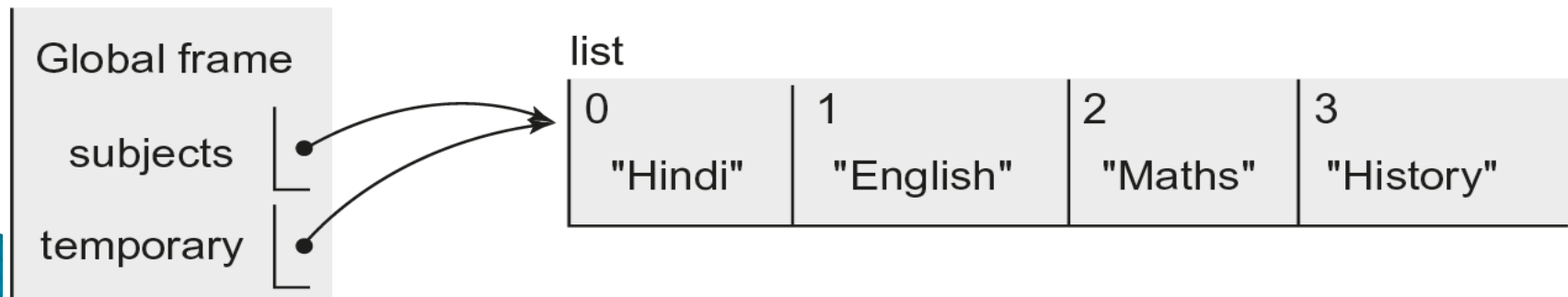
## Lists

- A list is an ordered sequence of values.
- Values stored in a list can be of any type such as string, integer, float, or list.
- Note!! Elements of a list are enclosed in square brackets, separated by commas.
- Unlike strings, lists are mutable, and therefore, one may modify individual elements of a list.

```
>>> subjects=['Hindi', 'English', 'Maths', 'History']
```



```
>>> temporary = subjects
```





```
>>>temporary[0] = 'Sanskrit'  
>>>print(temporary)  
>>>print(subjects)
```

```
['Sanskrit', 'English', 'Maths', 'History']
```

```
['Sanskrit', 'English', 'Maths', 'History']
```

```
>>>subjectCodes = [['Sanskrit', 43], ['English', 85], ['Maths', 65],  
['History', 36]]
```

```
>>>subjectCodes[1]
```

```
['English', 85]
```

```
>>>print(subjectCodes[1][0],subjectCodes[1][1])
```

```
English 85
```

## List Operations

```
>>>list1 = ['Red', 'Green']
```

```
list2 = [10, 20, 30]
```

## Multiple Operator \*

```
>>>list2 * 2
```

```
[10, 20, 30, 10, 20, 30]
```

## Concatenation Operator +

```
>>> list1 = list1 + ['Blue']
```

```
>>>list1
```

```
['Red', 'Green', 'Blue']
```



## Length Operator len

```
>>> len(list1)
```

```
3
```

## Indexing & Slicing

```
>>> list2[-1]
```

```
30
```

```
>>> list2[0:2]
```

```
[10, 20]
```

```
>>> list2[0:3:2]
```

```
[10, 30]
```

## Function min & max

```
>>> min(list2)
```

```
10
```

```
>>> max(list1)
```

```
'Red'
```

## Membership Operator: in

```
>>> 40 in list2
```

```
False
```

```
>>> students = ['Ram', 'Shyam', 'Gita', 'Sita']
```

```
for name in students:
```

```
    print(name)
```

A green rectangular box with a blue border containing the names Ram, Shyam, Gita, and Sita stacked vertically.

```
Ram  
Shyam  
Gita  
Sita
```



# Tuples and Dictionaries

## Tuples

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Note: Elements of a tuple can be possibly of heterogeneous types

```
Bright=('Red','Green','Blue')
```

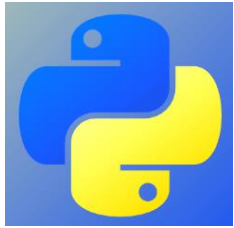
```
print(Bright)
```

```
Bright[1]='Orange' # Error
```

```
Light=('Pink','Grey') ('Red','Green','Blue')
```

```
Bright=Bright+Light ('Red','Green','Blue','Pink','Grey')
```

```
print(Bright) # No Error
```



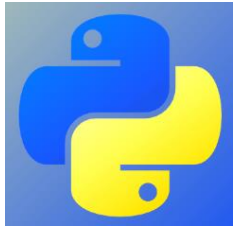
# Tuples can be represented with or without Paranthesis ( )

```
#Tuple
Values=10,12,9000
MValues=(11,12,9000)
NValues=11,12,9000
print(Values)
if MValues==NValues:
    print("Same to Same1")
else: print("No Same1")
TValues=12,11,9000
#Order of tuple matters
if MValues==TValues:
    print("Same to Same2")
else: print("No Same2")
```

```
Values=list(Values)
#Typecasting to List
Values[1]=909
print(Values)

MValues=sorted(MValues)
#Sorted tuple produces List
MValues[2]=899
print(MValues)
```

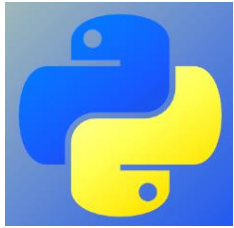
```
(10, 12, 9000)
Same to Same1
No Same2
[10, 909, 9000]
[11, 12, 899]
```



# Dictionary { }

A dictionary is a set of key: value pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}. Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.

```
Flat = { 'Mack': 140, 'Jack': 120 }
```



```
Flat = {'Mack': 104, 'Jack': 120}
```

```
print(Flat)
```

```
Flat['Raj']=132
```

```
Flat['Yog']=109
```

```
print(Flat)
```

```
del Flat['Jack']
```

```
print(Flat)
```

```
{'Mack': 104, 'Jack': 120}
```

```
{'Mack': 104, 'Jack': 120, 'Raj': 132, 'Yog': 109}
```

```
{'Mack': 104, 'Raj': 132, 'Yog': 109}
```

```
Mack Stays in Flat 104
```

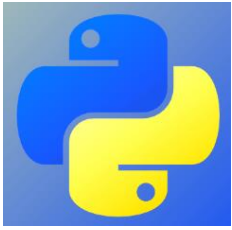
```
Raj Stays in Flat 132
```

```
Yog Stays in Flat 109
```

```
for k,v in Flat.items():
```

```
    print(k,'Stays in Flat',v)
```





# THANKS !!

Preeti Arora

[preetiarora.author@gmail.com](mailto:preetiarora.author@gmail.com)