Tuples in Python

A tuple is an ordered, immutable collection of elements.

Similar to lists, tuples can store multiple items of different data types, such as integers, floats, strings or even other tuples.

However, unlike lists, tuples cannot be modified once created, making them immutable.

Example of creating a tuple in Python:

```
my_tuple = (1, 2, 'a', 'b', True)
```

Characteristics of Tuples

1. **Immutable:**

Once a tuple is created, its elements cannot be modified. This immutability ensures the integrity of data stored in the tuple.

2. **Ordered:**

Tuples maintain the order of elements as they are defined, and this order is preserved throughout the tuple's lifetime.

~~A~~

③ Indexing and Slicing :

You can access individual elements of a tuple using indexing, similar to lists. Additionally, you can use slicing to extract a portion of a tuple.

Example :

```
my_tuple = (1,2,3,4,5)
print(my_tuple[0])  # Output : 1
print(my_tuple[1:4])  # Output : (2,3,4)
```

④ Tuple Packing and Unpacking :

Tuple packing involves creating a tuple by assigning multiple values to a single variable, while tuple unpacking involves assigning the values of a tuple to multiple variables.

Example :

```
# Tuple Packing
my_tuple = 1, 2, 'a'
print(my_tuple)  # Output : (1, 2, 'a')
```

```
# Tuple Unpacking
a, b, c = my_tuple
print(a) # Output 1
print(b) # Output 2
```

⑤ Size and Length :

You can determine the size or length of a tuple using the len() function.

Example :

```
my_tuple = (1, 2, 3, 4, 5)
size = len(my_tuple)
print(size)
```

Output : 5

Dictionaries in Python

A dictionary is a built-in data structure that allows you to store and retrieve data in key-value pairs.

It is an unordered collection of elements where each element is identified by a unique key.

Dictionaries are defined using curly braces {} and consist of key-value pairs separated by colons :.

Example of creating a dictionary in Python:

```
my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}
```

Characteristics of Dictionaries:

1. Mutable: Dictionaries are mutable, meaning you can modify, add, or remove key-value pairs after the dictionary is created.

2. Unordered: Dictionaries are unordered, which means the order of key-value pairs may vary and is not guaranteed.

The order of elements in a dictionary is based on the hash value of the keys.

(3) Accessing Values:

You can access values in a dictionary by referencing the corresponding key using square brackets [] or the get() method.

If the key does not exist, an error occurs when using square brackets, while get() returns None or a specified default value.

```
my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}
print(my_dict['name']) # Output : Alice
print (my_dict.get('age')) # Output :25
print (my_dict.get('gender')) # Output : None.
```

(4) Modifying and Adding Entries:

You can modify the value of an existing key or add new key-value pairs to a dictionary.

Example :

```
my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}
my_dict['age'] = 26 # Modifying an existing key value pair
my_dict['gender'] = 'Female' # Adding a new Key-value pair
print (my_dict)
```

⑤ Removing Entries :

You can remove entries from a dictionary using the del keyword or the pop() method, which removes and returns the value associated with a given key.

Example :

```
my_dict = {'name':'Alice', 'age':25, 'city':'New York'}
del my_dict['age']  # Removing a specific key-value pair
my_dict.pop('city') # Removing and returning the value of a key
print(my_dict)
```

Python List

A list is a built-in data structure that allows you to store and manipulate a collection of items.

Lists are ordered, mutable (changable), and can contain elements of different data types.

They are defined using square brackets [] and items are separated by commas.

Example of creating a list in Python:

```
my_list = [1, 2, 3, 'a', 'b', True]
```

Lists have several important characteristics and support various operations:

(1) Accessing Elements in Python List

You can access individual elements in a list using indexing.

Indexing starts from 0, so the first ~~item~~ element is at index 0, the second element is at index 1, and so on. Negative indexing allow you to access elements from the end of the list.

Ex:

```
my_list = [1, 2, 3, 4, 5]
print (my_list[0])        # output  1
print (my_list[-1])               5
```

(2) Modifying Elements in Python List

List are mutable, meaning you can change the value of individual elements or modify the list itself.

Example :

```
my_list = [1,2,3,4,5]
my_list[2] = 'a'
print(my_list)
```

Output ⇒ [1,2,'a',4,5]

```
my_list.append(6)
print(my_list)
```

Output ⇒ [1,2,'a',4,5,6]

(3) List Slicing in Python

Slicing allows you to extract a portion of a list. It is done by specifying a start index, an end index (exclusive), and an optional step size.

Example :
```
my_list = [1,2,3,4,5]
print(my_list[1:4]
```

Output : [2,3,4]

(4) List Operations in Python

List support various operations, including concatention, (+) repetition, (*) length calculation, len() membership testing and more.

Example :

```
list1 = [1,2,3]
list2 = [4,5,6]
Concatenated_list = list1 + list2
print (Concatenated_list)
```

Output ⇒ [1,2,3,4,5,6]

~~repeated_list1 * 3~~
~~print~~

```
repeated_list = list1 * 3
print (repeated_list)
```

Output ⇒ [1,2,3,1,2,3,1,2,3]

```
Length = len(my_list)
print (length)          # Output : 5
```

```
print (a in my_list)    # Output : True
```

⑤ List Methods in Python

List have several built-in methods that allow you to perform operations such as adding or removing elements, sorting, reversing and more.

Example :

```
my_list = [4,2,1,3]
my_list.append(5)
print my_list)        # Output: [4,2,1,3,5]

my_list.remove(2)
print (my_list)       # Output: [4,1,3,5]

my_list.sort()
print (my_list)       # Output: [1,3,4,5]
```

Function Body:

The function body contains the block of code that gets executed when the function is called.

It performs the desired task or computation. The body is indented under the function definition.

Example:

```
def greet(name):
    print("Hello, "+name+"!")
    # Additional code in the function body
    print("Welcome to our program!")
```

func body

Parameters (Arguments):

Parameters are placeholders for values that are passed into the function when it is called.

They allow you to provide input values to the function. Parameters are listed inside the parentheses in the function defination.

Example: `def greet(name):` ← Argument
`    print("Hello,"+name+"!")`

Return Statement (Optional):

The return statement is used to specify the value or values that the function should return as the result of its execution.

It is optional and if omitted, the function will return None by default.

Example :
```
def add_numbers(a,b):
    return a+b
```

Function Call :

The function call is where you invoke or execute the function with specific ~~arg~~ arguments (input values) to perform its task.

you use the function name followed by parentheses, with arguments (if any) inside the parentheses.

Example :
```
greet("Alice")
```

Function Arguments

Types of arguments that we can use to call a function:

Default arguments

Keyword arguments

Required arguments

Variable-length arguments

- **Default Arguments**

Default arguments allow you to assign default values to parameters in a function definition.

If an argument is not provided when calling the function, the default value is used.

Default arguments are helpful when you want to provide a common value that is used most of the time but can be overridden if needed.

Example :

```
def function(n1,n2=20):
print("number1 is :",n1)
print("number2 is :",n2)
print("Passing only one argument")
function(30)
print("Passing two arguments")
function(50,30)
```

O/P

```
Passing only one argument
number1 is : 30
number2 is : 20
Passing two arguments
number1 is : 50
number2 is : 30
```

Keyword Arguments

Keyword arguments allow you to specify arguments by their parameter names when calling a function, rather than relying on the order of arguments in the function definition. By using keywords, you can explicitly associate values with specific parameters, regardless of their position.

Example :

```
# Defining a Function

def function(n1,n2):
    print("number 1 is:",n1)
    print("number 2 is:",n2)

# Calling function and passing arguments without using keyword

print("Without using keyword")
function(50,30)

# Calling function and passing arguments using keyword

print("With using keyword")
function(n2=50, n1=30)
```

Output:

Without using keyword

number 1 is : 50
number 2 is : 30

With using Keyword

number 1 is : 30
number 2 is : 50

Add Two Numbers

```
# function that adds two numbers

def add_numbers (num1, num2):
    sum = num1 + num2
    return sum

# Calling function with two values

result = add_numbers (5,4)
print ('Sum:', result)
```

Output:

Sum : 9

Literal Constants :

Generally values assigned to variables are literal constants. For eg :- 4, 5, 4.6, 'x', "Hello". All are values

Numbers : Numeric value.

Integers - Whole Numbers like 7, 8, 9, 10 .....

Long Integers - Large whole Numbers like 456893281472

Floating Point Numbers - Decimal point like 2.3, 4.5.....

Complex Numbers - Form of $(x+yi)$ or $(4+5i)$ (x: real, yi: Imaginary)

Strings : Group of characters. Generally used for representing text.

```
>>> 'Hello' → o/p 'Hello'
>>> "Hello" → o/p 'Hello'
>>> '''Hello''' → o/p 'Hello'
```

Strings are immutable (once string is created it cannot be changed) in Python.

Ex ⇒
```
i = 10
print (i)
```
o/p → 10

```
str = "Hello"
print (str)
```
o/p → Hello

What is Identifier in Python?

(2) Variables and Identifiers :

Variables is a named location which is used to store data in a memory.

>>> rollno = 503 — Value assigned to variable
Variable name

Identifiers are the names given to identify something.

Identifiers can be assigned to → Variable
Class
Function
Module etc

Rules to Name Identifiers :

i) First character must be underscore ("_") or letter.

ii) Identifier names are case sensitive.

iii) Punctuation characters are not allowed.
→ @, $ and %.

E.g. _name = "LWN"
Print (_name)
o/p LWN —— valid.

1name = "LWN"
Print (1name)

o/p Syntax Error

3)

Data Types :

In Python, Variables can hold values of different types called datatypes.

In Python data types are actually classes and variables are instances of these classes.

Numbers :

Integer (int), Floating Point (float) and Complex numbers (Complex)

Classes

E.g: >>> x = 4 // int
>>> y = 5.0 // float
>>> z = 4+5j // complex

→ type() function : Returns class a variable or value belongs to

E.g : >>> type(x) → o/p (<Class, 'int'>)

↑ value, variablename

E.g: i = 10	i = 10.5
type (i)	type (i)
o/p int	o/p float

(4)

String : Sequence of unicode characters.

Eg: >>> S = "Python Programming"
print(S)

Comments : Non executable statements in code
marks beginning of comment.

[make letter readable Programs

```
# Comment
print ("Hello")
```

O/P Hello

Reserved words : These are the words with some predefined meaning.
Cannot be used for naming identifiers.

e.g : and class return try ....
not for import else .....
or if while except ....

Indentation : It is the whitespace at beginning of the line.

In Python it is used to associate and group statements.

[Statements in a block/group must have same indentation level]

⑤

Indentation

E.g:

```
x = 10
y = 5
if x > y :
    print ("x is large than y")
else :
    print ("y is large than x")
```

Single TAB Space for indentation

E.g ⇒

```
i = 10
if i > 10 :
    print ("A")
else :
    print ("B")
print ("C")
```

O/p ⇒ B
C

Taking input from user:

input() function is used.

prompts the user to provide some info/data.

Always take input as String. (if no. (for e.g. is entered it also treated as string.)

```
>>> roll_no = input ("Enter Roll No")
```

E.g:

```
i = input ("Enter the Number")
print (i)
print (type (i))
```

O/p ⇒ Enter the Number. 5 ← Enter value

5

<class 'str'> ← Printed

(6)

Operators : Are the constructs that are used to manipulate value of operands.

Arithmetic operators : +, -, *, /, %, **, and //.

print (5+7) [Addition] → o/p = 12
print (5-7) [Subtraction] → o/p = -2
print (5*7) [Multiplication] → o/p = 35
print (8/4) [Division] → o/p = 2

print (7 % 5) [Modulus] → o/p = 2 (Returns Remainder)

print (12//5) [Floor Division] → o/p = 2 (Returns Quotient)

print (10**20) [Exponent] → o/p = 10^{20}

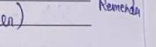
∵ Floor Division is a division operation that returns the largest integer that is less than or equal to the result of the division.

Examples : print (10/5) o/p 2.0

print (10//4) o/p 2 (Quotient)

print (10 % 4) o/p 2 (Remainder)

print (10**4) o/p 10000 (10^4)



f)

Comparison Operators :

Also called as Relational operators. [Returns True or False]

print (5==7) [EQUALITY] → o/p = False

print (5!=7) [NOT EQUAL TO] → o/p = True

print (5>7) [Greater than] → o/p = False

print (5<7) [Less than] → o/p = True

print (5>=7) [Greater than or Equal] → o/p = False

print (5<=7) [Less than or Equal] → o/p = True

Assignment and Shortcut operators :

Assignment operators assigns values to operand.

x = 5 (Assigns Value '5' to x)

x += 5 // x = x+5

x -= 5 // x = x-5

x *= 5 // x = x*5

? Same for all arithmetic operations ?

x /= 5 // x = x/5

eg – i = 10
i += 5
print (i)
o/p => 15

i = 10
i /= 5
print (i)
o/p => 2.0

(8)

Unary operators : Acts on single operands.

Unary minus (-) operator negates the value

```
x = 10
y = -(x)
    → Unary Minus (-)
```

Negates value of x (10) and assigns -10 to y.

e.g ⇒

```
i = 10
J = -(i)
print (J)
```

O/p ⇒ -10

Bitwise operators : Performs operations on Bit level.

⇒ AND (&) : If both bits are '1' then O/p '1' else '0'.

```
  1010
(AND) 0101
  0000 → Output
```

0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

⇒ OR (|) : If any or both bits '1' then O/p '1' else '0'

```
  1010
(OR) 0101
  1111 → Output
```

0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

(9)

⇒ NOT (~) : Unary operation '1' is converted to '0' and '0' to '1'.

~ 1011 → 0100 → O/p

⇒ XOR (^) : If bits are same in both integers then '0' else '1'.

```
  1011
  0101
  1110 → O/p
```

1 ^ 1 = 0
0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1

Shift operators : Used to shift bits to the left or to the right.

x = 1011
x << 1 → O/p = 0110

Left Shift operator] MSB is discarded, LSB is set to 0

MSB LSB
x = 1011
x >> 1 → O/p = 0101

Right Shift operator] MSB is set to 0, LSB is discarded

eg. i = 5	i = 5
print (i << i)	print (i >> 5)
O/p 10	O/p 2

(10)

Logical operators:

AND (&&) : If any two expressions are written like (Cond1) && (Cond2), then overall expression will return True if both conditions are True.

```
>>> (5<7) && (8>6) → True
       ↓          ↓
     True       True

>>> (5<7) && (8<6) → False
       ↓          ↓
     True       False
```

OR (||) : If any one condition is True, then output is True.

```
>>> (5<7) || (8<6) → True
       ↓          ↓
     True       False
```

NOT (!) : Negates value of the expression

⑪

Datatype Conversion :

Python Provides several in-built functions to convert a value from one data-type to another.

int(x), long(x), float(x), list(x) chr(x).....

SAMPLE PROGRAMS:

① Area of Triangle Using Heron's Formula.

```
x = float (input ("Enter First Side"))
```

Used to convert entered value (String) into float value.

```
y = float (input ("Enter Second Side"))
z = float (input ("Enter Third Side"))
s = (x+y+z)/2
area = (s*(s-x)*(s-y)*(s-z))**0.5
print (area)
```

exponent (Making Square Root)

② Area of Circle

```
radius = float (input ("Enter Radius"))
area = 3.14 * radius * radius
print (area)
```

(1)

An array can be

(2)

(3) Average of two numbers

```
n1 = int (input ("Enter First Number"))
n2 = int (input ("Enter Second Number"))
avg = (n1+n2)/2
print (avg)
```

O/P → Enter First Number 10
Enter Second Number 12
11.0

Python Comments

Comments are nothing but ignorable part of python program that are ignored by the interpreter. In ~~pthy~~ python we use # symbol to start writing a comment.
It enhance the readability of code.

There are 2-types of comments:

- Single line comments (#)
- Multi-line Comment (''') or (""") (''')

Comments are very important while writing a program. Commenting your code helps to explain your code thought process, and helps you and others to understand later on the intention of your code. You might forget the key details of the program you just wrote in a month's time. So taking the time to explain these concepts in the form of comments is always very fruitfull.

Ex: # This is a comment
# print out Hello
print('Hello')

Multiline Comments: We can have comment that extent up to multiple lines. One way is to use the has symbol at the beginning of each line.

Ex: # This is a long comment
and it extends
to multiple lines

Another way of doing this is to use triple quotes, either ''' or """

These triple quotes are generally used for multi line strings. But they can be used as a multi line comment as well.

Example: """ This is a perfect
Example of
multi line comment. """

''' This is also an example
for multi line comment '''

{Global variable}

When a variable is declared above a function, it becomes global variable.

These variables are available to all the function which are written after it

The scope of global variable is the entire program body written below it.

```
a = 50 <- Global Variable
def show():
    x = 10 <- Local variable
    print(a) <- Using Global variable inside func.
    print(x) <- Using Local variable inside function

show()
print("x:", x) <- Using Local variable outside func., it will show Error
print("a:", a) <- Using Global variable
```

O/P
```
50
10
a: 50
"x: is not defined error"
```

Python Variable

Variable is the name of memory location where we can stored different types of values.

⊙ type() :- It returns the data type of the given value

Ex : a = 10
print(a)
print(type(a))

O/P 10
<class 'int'>

Python List ....

a = [10, 'ankit', 10]
print(a)
print(type(a))

O/P [10, 'ankit', 10]
<class 'list'>

Allow duplicate value

Python Tuple ....

a = (10, 'ankush', 10)
print(a)
print(type(a))

O/P (10, 'ankush', 10)
<class 'tuple'>

Allow duplicate value.

Python Set ....

a = {23, 23, 'Akilish'}
print(a)
print(type(a))

O/P {'Akhilesh', 23}
<class 'set'>

Not allow duplicate value

- A variable is the name given to a memory location.
- Since Python is an infer language that is smart enough to determine the type of a variable, we do not need to specify its type in Python.
- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-Z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords.
- The value stored in a variable can be changed during program execution.
- A variables in Python is only a name given to a memory location, all the operations done on the variable effects that memory location.

{Local Variables}

The variable which are declared inside a function called as Local variable.

Local variables scope is limited only to that function where it is created. It means local variable value is available only in that function not outside of that function.

Keyword → def add(y): ← Function name, Parameter
    x = 10 ← Local variable
    print(x)] Using Local variable
    print(x+y)] inside function
add(20)
print(x) ← Using Local variable outside function, it will show error

O/P
10
30
Error x is not defined

①

Python is a general purpose dynamically typed, high level language developed by "Guido van Rossum" in the year 1991 at CWI in Netherland.

Syntax : print ("Hello");

* Input / Output :

- For input we have input()
- For output we have print()

Input () function :

In python programs, we use the input() function to take any kind of data input from the user through the keyboard.

The input() function waits for the user's input and as soon as the user enters the data and presses the enter key, the input() function reads the data and assigns it to the variable.

Syntax : var_name = input()

Input() function always return String value.

(2)

Example :- Python input() function

```
color = input("what color is rose?:")
print(color)

n = int(input("How many roses?:"))
print(n)

price = float(input("Price of each rose?:"))
print(price)
```

O/p
```
what color is rose?: red
red
How many roses?: 10
10
Price of each rose?: 15.50
15.5
```

Output function :

Printing output in Python is straightforward, thanks to the print() function.

Example. : print("Hello, World!")

O/p Hello, World!

print() Function - The print() function is used to print the specified message to the output screen/device. The message can be a string, or any other object.

Output Statements

print() function :

It is used to display the output on the Console.

The function allows us to display text, variables, and expressions on the console.

Syntax : print (objects, sep='character', end='character', file=sys.stdout, flush=False)

sep - Separate the objects by given character. Character can be any string. Default is ' ' or can write none.

end - It indicates ending character for the line. Default is '\n' or can write none.

file - An object with a write method. Default is sys.stdout or can write none.

flush - A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False.

print() - This function is used to display a blank line.

print ("String") - When a string is passed to the function, the string is displayed as it as.

print (object) - We can pass objects like list, tuples and dictionaries to display the elements of those objects.

Ex - data [10, 20, -50, 21.3, 'Hello'] ← [List]
print (data)

Output Statements

print ("string" sep=',') - It separates string with given sep character.
Character can be any string. Default is ' ' or can write none
Ex: print ("Like", "Share", sep='***')

O/P = Like *** Share

print ("string" end=',') - When ending character is passed. It prints given character at the end.
Default is '\n' or can write none.

Ex: print ("Welcome", end=' ') ← Space
print ("Home", end=' ')
print ("You", end='\t\n') For tab space
O/P: Welcome Home You

print (variable list): This is used to display the value of a variable or a list of variable.

Ex: x = 20 y = 30 print (x, y)	O/P 20 30 List of variable's value in the output screen, are separated by a space by default, we can change this by sep

print (x, y, sep=',')

Output: 20, 30

Output Statements

print ("String", variable list) –

This is used to display the string along with variable.

Ex: name = 'Tanu'

age = 62

print ("My Name is", name, "and My age is", age)

O/P : My Name is Tanu and My age is 62

Or

m = 40

print ("Value:", m)

O/P : Value : 40