**More programs:**

tuple1 = (14, 52, 17, 24)

print(tuple1[1])

print(tuple1[3])

tuple1 = (14, 52, 17, 24)

print( len(tuple1) )

tuple1 = (14, 52, 17, 24)

for item in tuple1:

print(item)

tuple1 = (14, 52, 17, 24)

index = 0

while index<len(tuple1):

print(tuple1[index])

index = index + 1

# Different types of tuples

# Empty tuple

my\_tuple = ()

print(my\_tuple)

# Tuple having integers

my\_tuple = (1, 2, 3)

print(my\_tuple)

# tuple with mixed datatypes

my\_tuple = (1, "Hello", 3.4)

print(my\_tuple)

# nested tuple

my\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

print(my\_tuple)

my\_tuple = 3, 4.6, "dog"

print(my\_tuple)

# tuple unpacking is also possible

a, b, c = my\_tuple

print(a) # 3

print(b) # 4.6

print(c) # dog

my\_tuple = ("hello")

print(type(my\_tuple)) # <class 'str'>

# Creating a tuple having one element

my\_tuple = ("hello",)

print(type(my\_tuple)) # <class 'tuple'>

# Parentheses is optional

my\_tuple = "hello",

print(type(my\_tuple)) # <class 'tuple'>

# Accessing tuple elements using indexing

my\_tuple = ('p','e','r','m','i','t')

print(my\_tuple[0]) # 'p'

print(my\_tuple[5]) # 't'

# IndexError: list index out of range

# print(my\_tuple[6])

# Index must be an integer

# TypeError: list indices must be integers, not float

# my\_tuple[2.0]

# nested tuple

n\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index

print(n\_tuple[0][3]) # 's'

print(n\_tuple[1][1]) # 4

Slicing operator is :

# Accessing tuple elements using slicing

my\_tuple = ('p','r','o','g','r','a','m','i','z')

# elements 2nd to 4th

# Output: ('r', 'o', 'g')

print(my\_tuple[1:4])

# elements beginning to 2nd

# Output: ('p', 'r')

print(my\_tuple[:-7])

# elements 8th to end

# Output: ('i', 'z')

print(my\_tuple[7:])

# elements beginning to end

# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

print(my\_tuple[:])

# Deleting tuples

my\_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# can't delete items

# TypeError: 'tuple' object doesn't support item deletion

# del my\_tuple[3]

# Can delete an entire tuple

del my\_tuple

# NameError: name 'my\_tuple' is not defined

print(my\_tuple)

# Membership test in tuple

my\_tuple = ('a', 'p', 'p', 'l', 'e',)

# In operation

print('a' in my\_tuple)

print('b' in my\_tuple)

# Not in operation

print('g' not in my\_tuple)

DICTIONARY------

# empty dictionary

my\_dict = {}

# dictionary with integer keys

my\_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys

my\_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()

my\_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair

my\_dict = dict([(1,'apple'), (2,'ball')])

# get vs [] for retrieving elements

my\_dict = {'name': 'Jack', 'age': 26}

# Output: Jack

print(my\_dict['name'])

# Output: 26

print(my\_dict.get('age'))

# Trying to access keys which doesn't exist throws error

# Output None

print(my\_dict.get('address'))

# KeyError

print(my\_dict['address'])

# Changing and adding Dictionary Elements

my\_dict = {'name': 'Jack', 'age': 26}

# update value

my\_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}

print(my\_dict)

# add item

my\_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}

print(my\_dict)

# Removing elements from a dictionary

# create a dictionary

squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# remove a particular item, returns its value

# Output: 16

print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}

print(squares)

# remove an arbitrary item, return (key,value)

# Output: (5, 25)

print(squares.popitem())

# Output: {1: 1, 2: 4, 3: 9}

print(squares)

# remove all items

squares.clear()

# Output: {}

print(squares)

# delete the dictionary itself

del squares

# Throws Error

print(squares)

# Membership Test for Dictionary Keys

squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: True

print(1 in squares)

# Output: True

print(2 not in squares)

# membership tests for key only not value

# Output: False

print(49 in squares)

# Dictionary Built-in Functions

squares = {0: 0, 1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: False

print(all(squares))

# Output: True

print(any(squares))

# Output: 6

print(len(squares))

# Output: [0, 1, 3, 5, 7, 9]

print(sorted(squares))

# Different types of sets in Python

# set of integers

my\_set = {1, 2, 3}

print(my\_set)

# set of mixed datatypes

my\_set = {1.0, "Hello", (1, 2, 3)}

print(my\_set)

# set cannot have duplicates

# Output: {1, 2, 3, 4}

my\_set = {1, 2, 3, 4, 3, 2}

print(my\_set)

# we can make set from a list

# Output: {1, 2, 3}

my\_set = set([1, 2, 3, 2])

print(my\_set)

# set cannot have mutable items

# here [3, 4] is a mutable list

# this will cause an error.

my\_set = {1, 2, [3, 4]}

# Distinguish set and dictionary while creating empty set

# initialize a with {}

a = {}

# check data type of a

print(type(a))

# initialize a with set()

a = set()

# check data type of a

print(type(a))

# initialize my\_set

my\_set = {1, 3}

print(my\_set)

# my\_set[0]

# if you uncomment the above line

# you will get an error

# TypeError: 'set' object does not support indexing

# add an element

# Output: {1, 2, 3}

my\_set.add(2)

print(my\_set)

# add multiple elements

# Output: {1, 2, 3, 4}

my\_set.update([2, 3, 4])

print(my\_set)

# add list and set

# Output: {1, 2, 3, 4, 5, 6, 8}

my\_set.update([4, 5], {1, 6, 8})

print(my\_set)

**LAMBDA:**

#lambda function

square = lambda a: a\*a

#call lambda function

result = square(6)

print(result)

#lambda function

mul = lambda a,b: a\*b

#call lambda function

result = mul(5,3)

print(result)

#lambda function

six = lambda : 6

#call lambda function

result = six()

print(result)

#recursive lambda function

factorial = lambda a: a\*factorial(a-1) if (a>1) else 1

#call lambda function

result = factorial(5)

print(result)

import math

#function returning lambda function

def myfunc(n):

return lambda a : math.pow(a, n)

#lambda functions

square = myfunc(2) #square = lambda a : math.pow(a, 2)

cube = myfunc(3) #cube = = lambda a : math.pow(a, 3)

squareroot = myfunc(0.5) #squareroot = lambda a : math.pow(a, 0.5)

print(square(3))

print(cube(3))

print(squareroot(3))

Program to double each item in a list using map()

my\_list = [1, 5, 4, 6, 8, 11, 3, 12]

new\_list = list(map(lambda x: x \* 2 , my\_list))

print(new\_list)