"""

syntax of function

def <name of function>(<function's argument which can be , separated>):

pass

"""

# eg 1: how to define a func

# def nothing(parameters, arguments):

# """ docstring - it list what func does """

# # body of the functions

# pass

# eg 2: func are defined before calling

# def greet(name):

# """ Given an input name this func will print the greeting for the person."""

# print("Hello " + name + ", Good Evening!")

#

#

# greet("Ajay")

# eg: 3: calling func before declaration

# greet\_a\_person("Ajay") # this will raise an error as call is before definition

# def greet\_a\_person(name):

# """ Given an input name this func will print the greeting for the person."""

# print("Hello " + name + ", Good Evening!")

""" return statements in functions """

# eg : 1

# def add(a, b):

# c = a + b

# return c

#

#

# x = add(3, 5)

# print(x)

# eg: 2 more than one value return

# def add\_two(a, b):

# c = a + b

# d = b - a

# return c, d

#

#

# x, y = add\_two(3, 5)

# print(x, y)

# print(type(x), type(y))

# eg: 3 returns a value as a tuple

# def add\_two(a, b):

# c = a + b

# d = b - a

# return c, d

#

#

# x = add\_two(3, 5)

# print(x)

# print(type(x))

# eg: 4 returns a value as a list

# def add\_two(a, b):

# c = a + b

# d = b - a

# return [c, d]

#

#

# x = add\_two(3, 5)

# print(x)

# print(type(x))

# eg: 5 returns a value as a dict

# def add\_two(a, b):

# c = a + b

# d = b - a

# return {"sum": c, "diff": d}

#

#

# x = add\_two(3, 5)

# print(x)

# print(type(x))

""" having more than one return """

# def is\_number\_positive(num):

# """ it determines whether a given num is postive or not and return True or False """

# if num > 0:

# return True

# else:

# return False

#

#

# print(is\_number\_positive(-2))

""" scope of variables in a func"""

# def my\_func():

# """ it just print the value of a"""

# a = 5 # local variable for a func

# print("the value of a inside a func is ", a)

#

#

# a = 20 # global variable for entire file

# my\_func()

# print("value of a outside of func ", a)

#

# """ calling a func inside another function"""

#

#

# def call\_my\_func():

# """ this will call another func """

# print("I am in call\_my\_func")

# my\_func()

#

#

# def func\_2():

# print("I am in func\_2 func")

# call\_my\_func()

#

#

# func\_2()

# using global variable in side a func and any update on global variable will update the value globally

# var\_global = 20

#

#

# def check\_global\_value():

# global var\_global # global keywords will give value of global variable

# print("seq 2: after using global keyword ", var\_global)

# var\_global = var\_global + 10

# print("seq 3: after addition operation", var\_global)

#

#

# print("seq 1: ", var\_global)

# check\_global\_value()

# print("seq 4: coming back to global level", var\_global)

""" learning arguments of a func"""

# def day\_greeting(f\_name, m\_name="m\_name", l\_name="lname", message=""):

# """ this will print the greeting message for a person name"""

# print("Hello " + f\_name + ", " + message)

# print("Hello " + f\_name + ", " + m\_name + " " + l\_name + " " + message)

#

#

# # eg: 1 passing both as positional argument

#

# day\_greeting("Manoj", "Good Morning!", "hey", "bye")

#

# # eg: 2 - passing as positional argument and other as keyword argument

#

# day\_greeting("Manoj", message="Good Morning!")

""" function with no default value as parameters"""

# def hello\_greeting(f\_name, m\_name, l\_name, message):

# """ this will print the greeting message for a person name"""

# print(f\_name)

# print(m\_name)

# print(l\_name)

# print(message)

#

#

# # eg: 1- calling with all positional argument

#

# hello\_greeting("m", "k", "s", "bye")

#

# # eg 2 : only few positional argument specified while calling

#

# hello\_greeting("m", "k") # TypeError: hello\_greeting() missing 2 required positional arguments: 'l\_name' and 'message'

#

""" Arbitrary Arguments"""

# def eve\_greeting(\*names):

# """ this will print the greeting message for a person name"""

# print(type(names)) # this will generate tuple

# for name in names:

# print("Hello, " + name)

#

#

# eve\_greeting("a", "b", "c", "d", "e", "f")

def afternoon\_greeting(names):

""" this will print the greeting message for a person name"""

print(type(names)) # this will generate tuple

for name in names:

print("Hello, " + name)

afternoon\_greeting(["a", "b", "c", "d", "e", "f"])