# -\*- coding: utf-8 -\*-

"""

Created on Fri Nov 8 11:52:47 2019

@author: sugandha.sharma

"""

"""Find length of a string """

str1=input("enter String")

count = 0

for char in str1:

count += 1

print(count)

"""Find length of a string using function"""

def string\_length(str1):

count = 0

for char in str1:

count += 1

return count

print(string\_length('Sugandha '))

"""Write a Python program to count the number of

characters(character frequency) in a string.

Sample String : google.com'

Expected Result : {'o': 3, 'g': 2, '.': 1, 'e': 1, 'l': 1,

'm': 1, 'c': 1}"""

def char\_frequency(str1):

dict = {}

for n in str1:

keys = dict.keys()

if n in keys:

dict[n] += 1

else:

dict[n] = 1

return dict

print(char\_frequency('google.com'))

"""PYTHON FUNCTIONS"""

"""A function is a block of code which only runs when

it is called.You can pass data, known as parameters,

into a function.

A function can return data as a result."""

"""Creating a Function:

In Python a function is defined using the def keyword:"""

def my\_function():

print("Hello from a function")

"""Calling a Function:

To call a function, use the function name followed by

parenthesis: """

def my\_function1():

print("Hello from a function")

my\_function1()

"""Parameters:

Information can be passed to functions as parameter.

Parameters are specified after the function name,

inside the parentheses. You can add as many parameters

as you want, just separate them with a comma.

The following example has a function with one

parameter (fname). When the function is called,

we pass along a first name, which is used inside

the function to print the full name:"""

def my\_function2(fname):

print(fname + " Sharma")

my\_function2("Anil")

my\_function2("Tina")

my\_function2("Raghav")

"""Default Parameter Value:

The following example shows how to use a default

parameter value.

If we call the function without parameter,

it uses the default value:"""

def my\_function3(country = "Norway"):

print("I am from " + country)

my\_function3("Sweden")

my\_function3("India")

my\_function3()

my\_function3("Brazil")

"""Passing a List as a Parameter:

You can send any data types of parameter to a

function (string, number, list, dictionary etc.),

and it will be treated as the same data type inside

the function.

E.g. if you send a List as a parameter,

it will still be a List when it reaches the function:"""

fruits = ["apple", "banana", "cherry"]

def my\_function4(food):

for x in food:

print(x)

#fruits = ["apple", "banana", "cherry"]

my\_function4(fruits)

"""

Return Values:

To let a function return a value,

use the return statement:

The phrase Keyword Arguments are often

shortened to kwargs in Python documentations.

"""

def my\_function5(x):

return 5 \* x

print(my\_function5(3))

print(my\_function5(5))

print(my\_function5(9))

"""

Arbitrary Arguments:

If you do not know how many arguments that

will be passed into your function,

add an \* before the parameter name in the

function definition.

This way the function will receive a tuple of

arguments, and can access the items accordingly:

Example:

If the number of arguments are unknown,

add a \* before the parameter name:"""

def my\_function6(\*kids):

print("The youngest child is " + kids[2])

my\_function6("anil", "raghav", "Tina","sunita ","Manika ")

"""

The pass Statement:

function definitions cannot be empty,

but if you for some reason have a function

definition with no content, put in the pass

statement to avoid getting an error.

"""

def myfunction7():

pass

"""Recursion """

def calc\_factorial(x):

if x == 1:

return 1

else:

return (x \* calc\_factorial(x-1))

num=4

print("The factorial of", num, "is", calc\_factorial(num))

"""Pass by reference vs value

All parameters (arguments) in the Python language are

passed by reference. It means if you change what a

parameter refers to within a function, the change also

reflects back in the calling function.

For example −

Function definition is here:"""

def changeme( mylist ):

"This changes a passed list into this function"

mylist.append([1,2,3,4])

print ("Values inside the function: ", mylist)

return

mylist = [10,20,30]

changeme( mylist )

print ("Values outside the function: ", mylist)

"""Here, we are maintaining reference of the passed

object and appending values in the same object.

So, this would produce the following result −

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]

Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

There is one more example where argument is being passed

by reference and the reference is being overwritten

inside the called function."""

def changeme( mylist ):

"This changes a passed list into this function"

mylist = [1,2,3,4]; #This would assign new reference

#in mylist

print( "Values inside the function: ", mylist)

return

mylist = [10,20,30]

changeme( mylist )

print ("Values outside the function: ", mylist)

"""The parameter mylist is local to the function

changeme. Changing mylist within the function does

not affect mylist. The function accomplishes nothing

and finally this would produce the following result −

Values inside the function: [1, 2, 3, 4]

Values outside the function: [10, 20, 30]

"""

"""A lambda function is a small anonymous function.

A lambda function can take any number of arguments,

but can only have one expression.

In Python, anonymous function is a function that is

defined without a name.

While normal functions are defined using the def

keyword, in Python anonymous functions are defined

using the lambda keyword.

Hence, anonymous functions are also called lambda

functions.

Syntax:

lambda arguments : expression

The expression is executed and the result is returned:

"""

"""A lambda function that adds 20 to the number

passed in as an argument, and print the result:"""

x = lambda a : a + 20

print(x(5))

"""Lambda functions can take any number of arguments:

Example:

A lambda function that multiplies argument a with

argument b and print the result:"""

x = lambda a, b : a \* b

print(x(3, 10))

"""

Example

A lambda function that sums argument a, b, and c

and print the result:"""

x = lambda a, b, c : a + b + c

print(x(10, 20, 30))

"""Why Use Lambda Functions?

The power of lambda is better shown when

you use them as an anonymous function inside another

function.

Say you have a function definition that takes one

argument, and that argument will be multiplied with an

unknown number:

"""

def myfunc(n):

return lambda a : a \* n

"""Use that function definition to make a function that

always doubles the number you send in:

"""

def myfunc1(n):

return lambda a : a \* n

mydoubler = myfunc1(2)

print(mydoubler(11))

"""

Or, use the same function definition to make a

function that always triples the number you send in:

"""

def myfunc(n):

return lambda a : a \* n

mytripler = myfunc(3)

print(mytripler(11))

"""Or, use the same function definition to make both

functions, in the same program:"""

def myfunc(n):

return lambda a : a \* n

mydoubler = myfunc(2)

mytripler = myfunc(3)

print(mydoubler(11))

print(mytripler(11))

"""Use lambda functions when an anonymous function

is required for a short period of time."""