Python Exercises- Functions

<https://www.w3resource.com/python-exercises/python-functions-exercises.php>

#1. Write a Python function to find the maximum of three numbers.

def mo3(n1,n2,n3):

return max((n1,n2,n3))

print(mo3(4,7,3))

#Textbook

def max\_of\_two( x, y ):

if x > y:

return x

return y

def max\_of\_three( x, y, z ):

return max\_of\_two( x, max\_of\_two( y, z ) )

print(max\_of\_three(3, 6, -5))

'''

2. Write a Python function to sum all the numbers in a list.

Sample List : (8, 2, 3, 0, 7)

Expected Output : 20

'''

def sums(lsts):

return sum(lsts)

print(sums((8,2,3,0,7)))

#Textbook

def sum(numbers):

total = 0

for x in numbers:

total += x

return total

print(sum((8, 2, 3, 0, 7)))

'''

3. Write a Python function to multiply all the numbers in a list.

Sample List : (8, 2, 3, -1, 7)

Expected Output : -336

'''

def mul(lsts):

res=1

for i in lsts:

res\*=i

return res

print(mul((8,2,3,-1,7)))

'''

4. Write a Python program to reverse a string.

Sample String : "1234abcd"

Expected Output : "dcba4321"

'''

def rs(strs):

lst1=list(strs)

lst1.reverse()

return ''.join(lst1)

print(rs('1234abcd'))

#Textbook

def string\_reverse(str1):

rstr1 = ''

index = len(str1)

while index > 0:

rstr1 += str1[ index - 1 ]

index = index - 1

return rstr1

print(string\_reverse('1234abcd'))

'''

5. Write a Python function to calculate the factorial of a number

(a non-negative integer). The function accepts the number as an argument

'''

def factorial(n):

res=1

for i in range(2,n+1):

res\*=i

return res

print(factorial(5))

#Textbook

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n-1)

'''

6. Write a Python function to check whether a number falls

within a given range

'''

def fwgr(num,ll,ul):

return num in range(ll,ul+1)

print(fwgr(55,18,100))

print(fwgr(3,10,20))

#Textbook

def test\_range(n):

if n in range(3,9):

print( " %s is in the range"%str(n))

else :

print("The number is outside the given range.")

test\_range(5)

'''

7. Write a Python function that accepts a string and counts the number

of upper and lower case letters.

Sample String : 'The quick Brow Fox'

Expected Output :

No. of Upper case characters : 3

No. of Lower case Characters : 12

'''

def noulc(strs):

nouc=len([i for i in strs if i.isupper()])

nolc=len([i for i in strs if i.islower()])

return 'upper and lower case no.:', nouc,nolc

print(noulc('The quick Brow Fox')) #('upper and lower case no.:', 3, 12)

#Textbook

def string\_test(s):

d={"UPPER\_CASE":0, "LOWER\_CASE":0}

for c in s:

if c.isupper():

d["UPPER\_CASE"]+=1

elif c.islower():

d["LOWER\_CASE"]+=1

else:

pass

print ("Original String : ", s)

print ("No. of Upper case characters : ", d["UPPER\_CASE"])

print ("No. of Lower case Characters : ", d["LOWER\_CASE"])

string\_test('The quick Brown Fox')

'''

8. Write a Python function that takes a list and returns a new list

with distinct elements from the first list.

Sample List : [1,2,3,3,3,3,4,5]

Unique List : [1, 2, 3, 4, 5]

'''

def de(lsts):

return list(set(lsts))

print(de([1,2,3,3,3,3,4,5]))

#Textbook

def unique\_list(l):

x = []

for a in l:

if a not in x:

x.append(a)

return x

print(unique\_list([1,2,3,3,3,3,4,5]))

'''

9. Write a Python function that takes a number as a parameter and checks whether the number is prime or not. Go to the editor

Note : A prime number (or a prime) is a natural number greater than 1 and that has no positive divisors other than 1 and itself.

'''

#Textbook

def test\_prime(n):

if (n==1):

return False

elif (n==2):

return True;

else:

for x in range(2,n):

if(n % x==0):

return False

return True

print(test\_prime(9))

'''

10. Write a Python program to print the even numbers from a given list.

Sample List : [1, 2, 3, 4, 5, 6, 7, 8, 9]

Expected Result : [2, 4, 6, 8]

'''

def even(lsts):

return [i for i in lsts if i%2==0]

print(even( [1, 2, 3, 4, 5, 6, 7, 8, 9]))

#Textbook

def is\_even\_num(l):

enum = []

for n in l:

if n % 2 == 0:

enum.append(n)

return enum

print(is\_even\_num([1, 2, 3, 4, 5, 6, 7, 8, 9]))

'''

11. Write a Python function to check whether a number is "Perfect" or not.

According to Wikipedia : In number theory, a perfect number is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself (also known as its aliquot sum). Equivalently, a perfect number is a number that is half the sum of all of its positive divisors (including itself).

Example : The first perfect number is 6, because 1, 2, and 3 are its proper positive divisors, and 1 + 2 + 3 = 6. Equivalently, the number 6 is equal to half the sum of all its positive divisors: ( 1 + 2 + 3 + 6 ) / 2 = 6. The next perfect number is 28 = 1 + 2 + 4 + 7 + 14. This is followed by the perfect numbers 496 and 8128.

'''

#Textbook

def perfect\_number(n):

sum = 0

for x in range(1, n):

if n % x == 0:

sum += x

return sum == n

print(perfect\_number(6))

'''

12. Write a Python function that checks whether a passed string is

a palindrome or not.

Note: A palindrome is a word, phrase, or sequence that reads the same backward as forward, e.g., madam or nurses run.

'''

#Textbook

def isPalindrome(string):

left\_pos = 0

right\_pos = len(string) - 1

while right\_pos >= left\_pos:

if not string[left\_pos] == string[right\_pos]:

return False

left\_pos += 1

right\_pos -= 1

return True

print(isPalindrome('aza'))

'''

13. Write a Python function that prints out the first n rows

of Pascal's triangle.

Note : Pascal's triangle is an arithmetic and geometric figure first imagined by Blaise Pascal.

Sample Pascal's triangle :

Pascal's triangle

Each number is the two numbers above it added together

'''

def pascal(n):

if n<1: print('row-number must be greater than 0')

elif n==1: print([1])

else:

res=[[1]]

print([1])

for i in range(2,n+1):

last=res[-1]

last.insert(0,0)

last.insert(len(last),0)

new=[last[j]+last[j+1] for j in range(len(last)-1)]

res.append(new)

print(res[-1])

pascal(3)

pascal(8)

pascal(0)

pascal(1)

#Textbook

def pascal\_triangle(n):

trow = [1]

y = [0]

for x in range(max(n,0)):

print(trow)

trow=[l+r for l,r in zip(trow+y, y+trow)]

return n>=1

pascal\_triangle(6)

'''

14. Write a Python function to check whether a string is a pangram or not.

Note : Pangrams are words or sentences containing every letter of the alphabet at least once.

For example : "The quick brown fox jumps over the lazy dog"

'''

def is\_pangram(strs):

lstrs=strs.lower()

db='abcdefghijklmnopqrstuvwxyz'

return all([(i in lstrs) for i in db])

print(is\_pangram( "The quick brown fox jumps over the lazy dog"))

print(is\_pangram("the important thing is to practice piano every day!"))

#Textbook

import string, sys

def ispangram(str1, alphabet=string.ascii\_lowercase):

alphaset = set(alphabet)

return alphaset <= set(str1.lower())

print ( ispangram('The quick brown fox jumps over the lazy dog'))

'''

15. Write a Python program that accepts a hyphen-separated sequence

of words as input and prints the words in a hyphen-separated sequence

after sorting them alphabetically.

Sample Items : green-red-yellow-black-white

Expected Result : black-green-red-white-yellow

'''

def shss(strs):

return '-'.join(sorted(strs.split(sep='-')))

print(shss('green-red-yellow-black-white'))

#Textbook

items=[n for n in input().split('-')]

items.sort()

print('-'.join(items))

'''

16. Write a Python function to create and print a list where the values

are the squares of numbers between 1 and 30 (both included).

'''

print([i\*\*2 for i in range(1,31)])

#Textbook

def printValues():

l = list()

for i in range(1,21):

l.append(i\*\*2)

print(l)

printValues()

'''

17. Write a Python program to create a chain of function decorators

(bold, italic, underline etc.).

'''

#Textbook

def make\_bold(fn):

def wrapped():

return "<b>" + fn() + "</b>"

return wrapped

def make\_italic(fn):

def wrapped():

return "<i>" + fn() + "</i>"

return wrapped

def make\_underline(fn):

def wrapped():

return "<u>" + fn() + "</u>"

return wrapped

@make\_bold

@make\_italic

@make\_underline

def hello():

return "hello world"

print(hello()) ## returns "<b><i><u>hello world</u></i></b>"

'''

decorator exercises

1. Create a decorator that times how long a function takes to run

'''

import time

#warmup

print(time.perf\_counter())

def timer(func):

def wrapper(\*args, \*\*kwargs):

start\_time = time.perf\_counter()

result = func(\*args, \*\*kwargs)

end\_time = time.perf\_counter()

total\_time = end\_time - start\_time

print(f"Function {func.\_\_name\_\_} took {total\_time:.6f} seconds to run.")

return result

return wrapper

@timer

def my\_func(order):

cnt=0

for i in range(10\*\*order):

cnt+=1

return cnt

print(my\_func(4))

'''

2. Create a decorator that logs the arguments and return value of a function

'''

import functools

import logging

logging.basicConfig(level=logging.DEBUG)

def log(func):

@functools.wraps(func)

def wrapper(\*args, \*\*kwargs):

print("Inside wrapper function")

logger = logging.getLogger(func.\_\_module\_\_)

logger.info(f"Calling {func.\_\_name\_\_} with args: {args}, kwargs: {kwargs}")

result = func(\*args, \*\*kwargs)

logger.info(f"{func.\_\_name\_\_} returned {result}")

return result

return wrapper

logging.basicConfig(level=logging.INFO)

# or ' logging.basicConfig(level=logging.DEBUG) ''

@log

def my\_function(arg1, arg2):

print('my func here:')

print(arg1+arg2)

my\_function('hello','mick')

#18. Write a Python program to execute a string containing Python code

str1="print('hello world')"

str2='''

for i in range(5):

print(i)

'''

exec(str1)

exec(str2)

#Textbook

mycode = 'print("hello world")'

code = """

def mutiply(x,y):

return x\*y

print('Multiply of 2 and 3 is: ',mutiply(2,3))

"""

exec(mycode)

exec(code)

#19. Write a Python program to access a function inside a function

def outer(base):

n=int(input('input the power-number:\n'))

def inner(power):

return base\*\*power

return 'the result is:', inner(n)

print(outer(5))

#Textbook

def test(a):

def add(b):

nonlocal a

a += 1

return a+b

return add

func= test(4)

print(func(4)) #9

'''

20. Write a Python program to detect the number of local variables

declared in a function.

Sample Output:

3

'''

#Textbook

def abc():

x = 1

y = 2

str1= "w3resource"

print("Python Exercises")

print(abc.\_\_code\_\_.co\_nlocals)

#warmup

def my\_func(a, b, c):

x = 1

y = 2

z = 3

return a + b + c

print(my\_func.\_\_code\_\_.co\_nlocals) # Output: 6

def my\_func(a, b, c):

return a + b + c

print(my\_func.\_\_code\_\_.co\_argcount) # Output: 3

def my\_func(a, b, c):

return a + b + c

print(my\_func.\_\_code\_\_.co\_varnames[:my\_func.\_\_code\_\_.co\_argcount]) # Output: ('a', 'b', 'c')

#use slicing to get only the argument names from co\_varnames. We slice from index 0 to co\_argcount to get only the argument names and exclude any local variable names

#nonlocal: using nonlocal keyword in Python is when we want to access and modify a variable in the enclosing scope of a nested function. In this case, we use the nonlocal keyword instead of global.

a = 1

def f():

b = 6

c = 7

def f1():

nonlocal c

c = 20

f1()

print("non local updated value:", c)

f() #non local updated value: 20

'''

21. Write a Python program that invokes a function after

a specified period of time.

Sample Output:

Square root after specific miliseconds:

4.0

10.0

158.42979517754858

'''

#warmup

from time import sleep

print('here')

sleep(0.03)

print('comes')

sleep(0.03)

print('the Hammer!')

#Textbook

from time import sleep

import math

def delay(fn, ms, \*args):

sleep(ms / 1000)

return fn(\*args)

print("Square root after specific miliseconds:")

print(delay(lambda x: math.sqrt(x), 100, 16))

print(delay(lambda x: math.sqrt(x), 1000, 100))

print(delay(lambda x: math.sqrt(x), 2000, 25100))