**Python Test**

1. **Question**

### GRADED

### Build a function called 'first\_starting\_vowel'

### ACCEPT a list of strings as input

### RETURN the first string that starts with a lowercase vowel ("a","e","i","o",or "u")

### HOWEVER if no string starts with vowel, RETURN the empty string ("")

### YOUR ANSWER BELOW

def first\_starting\_vowel(string\_list):

"""

Return the first string in the list that starts with a lowercase vowel

Positional Argument:

string\_list -- a list of strings

Example:

example\_list = ["hello","these","are","strings","in","a","list"]

print(first\_starting\_vowel(example\_list)) # --> "are"

print(first\_starting\_vowel(example\_list[:2]))#--> ""

"""

for string in string\_list:

return string

1. Response

string\_list = [x for x in input().split()]

def first\_starting\_vowel(string\_list):

A=[‘a’,’e’,’i’,’o’,’u’]

for string in string\_list:

If string[0] in A:

Return string

Else:

Return ‘ ‘

1. **Question**

### GRADED

### Code a function called 'halve\_to\_2'

### ACCEPT one numeric input.

### If the number <= 0, RETURN -1

### If the number > 0, divide that integer over-and-over by 2 until it becomes smaller than 2.

### RETURN that smaller-than-2 number

### e.g. input of 4 Will yield 1 (4->2->1), 5 yields 1.25 (5->2.5->1.25) etc.

### YOUR ANSWER BELOW

def halve\_to\_2( num ):

"""

Divide input-number by 2 until it becomes smaller than 2, then return.

If input-number <=0 return -1

Positional Argument:

num -- numeric input

Example:

print(halve\_to\_2(4)) #--> 1

print(halve\_to\_2(-39)) #--> -1

print(halve\_to\_2)(5673) #--> 1.385009765625

"""

return num

1. Response

N=int(input())

Def halve\_to\_2(num):

If num <= 0:

Return 0

Else:

While True:

num=num/2

If num < 2:

Return num

halve\_to\_2(4)

1. **Question**

### GRADED

### Code a function called 'string\_exansion'

### ACCEPT a non-empty string as input

### RETURN a string that contains every other character, 2n+2 times, where n is the original index of the letter.

### e.g. Input of "Hello" should result in "HHlllllloooooooooo".

### Input of "ROBErt" should result in "RRBBBBBBrrrrrrrrrr"

### YOUR ANSWER BELOW

def string\_expansion( input\_string ):

"""

Given a string input, return a string containing every other character 2n+2 times

Where "n" is the 0-based index of the letter

Positional Argument:

input\_string -- a non-empty string

Example:

str1 = "Hello"

str2 = "naME"

print(string\_expansion(str1)) #--> HHlllllloooooooooo

print(string\_expansion(str2)) #--> nnMMMMMM

"""

Pass

1. Response

Srting= input()

n=2

Index=0

Ret\_str=’’

Def string\_expansion(string)

If String:

Str=string.split()

Ret\_str+= Str[index]\*n

Else:

Print(‘string is not given !’)

1. **Question**

### GRADED

### Code a function called 'item\_count\_from\_index'

### ACCEPT two inputs, a list and an integer-index

### RETURN a count (number) of how many times the item at that index appears in the list.

### HOWEVER, if the integer-index is out of bounds for the list RETURN the empty string ("")

### (e.g. list of 3 items, index of 5 is out of bounds)

### YOUR ANSWER BELOW

def item\_count\_from\_index( input\_list, index):

"""

Return the count of items in a list found at a certain index

If index out of bounds, RETURN ""

Positional Argument:

input\_list -- a list of items, of unspecified types,

assume items are comparable. e.g. support == and != comparison

index -- an integer index

Examples:

print(item\_count\_from\_index([1,2,2,3,3,2,4],2)) #--> 3

print(item\_count\_from\_index([],2)) #--> ""

"""

Pass

1. Response
2. **Question**

### GRADED

### Code a function called 'length\_times\_largest'

### ACCEPT a list as input

### RETURN the length of the list times the largest integer (not float) in the list

### HOWEVER if the list does not contain an integer, RETURN the empty string ("")

### YOUR ANSWER BELOW

def length\_times\_largest(input\_list):

"""

Given a list of objects, return the length of the list times the

largest integer in the list.

Positional Argument:

input\_list -- a list of objects of unspecified types.

Example:

print(length\_times\_largest([1,2,3,4])) #--> 16

print(length\_times\_largest(["a","b","c",4])) #--> 16

print(length\_times\_largest(["1","100",2])) #--> 6

print(length\_times\_largest(["a","b"])) #--> ""

print(length\_times\_largest([0.0,40.6])) #--> ""

"""

Pass

1. Response
2. **Question**

### GRADED

### Code a function called 'combine'

### ACCEPT two inputs:

### The first input is a list.

### The second input is either either a list or some other type of object

### IF AND ONLY IF the second input is a \*list\*;

### ### "extend" the first list by adding to it the elements of the second list.

### e.g. if the inputs are [1,2,3], [4,5], the output should be [1,2,3,4,5] NOT [1,2,3,[4,5]].

### IF the second input is NOT a list, append that item to the original list.

### e.g. if the inputs are [1,2,3], (4,5), the output should be [1,2,3,(4,5)] NOT [1,2,3,4,5].

### RETURN the resulting combination.

### YOUR ANSWER BELOW

def combine(list1, to\_add):

"""

Return the combination of the two inputs

Positional Argument:

list1 -- a list of objects

to\_add -- an object, list or otherwise

Example:

l1 = [1,2,3]

a1 = [4,5]

a2 = "b"

a3 = (2,"b") # a tuple

print(combine(l1,a1)) #-->[1,2,3,4,5]

print(combine(l1,a2)) #-->[1,2,3,'b']

print(combine(l1,a3)) #-->[1,2,3,(2,'b')]

"""

Pass

1. Response
2. **Question**

### GRADED

### Code a function called 'type\_and\_length'

### ACCEPT one input of any of the six types used thus far (int, str, float, bool, list, tuple)

### RETURN a 3-tuple of (input, type, length)

### Hint: type can be found with type(<input>)

### If the <input> does not have `len()` return None for length.

### Note, None is not a string, it is a type and object in and of itself.

print(type(None)) # None example

### YOUR ANSWER BELOW

def type\_and\_length(obj):

"""

Return a tuple containing the inputted object, its type and length

Positional Argument:

obj - an object that may be any of the following:

(int, str, float, bool, list, tuple)

Example:

for obj in (1,"hi",1.5, True, [1,2,3],(4,5)):

print(type\_and\_length(obj))

#-->

(1, <class 'int'>, None)

('hi', <class 'str'>, 2)

(1.5, <class 'float'>, None)

(True, <class 'bool'>, None)

([1, 2, 3], <class 'list'>, 3)

((4, 5), <class 'tuple'>, 2)

"""

pass

1. Response
2. **Question**

### GRADED

### Code a function called 'reverse\_zip'

### ACCEPT a zip object

### ### The zip object will be in the exact same format as the above example (different values)

### RETURN a list of the 3 lists passed to 'zip()'

### e.g. above the function would return [[1,2],[3,4],[5,6]] for z\_obj above

### YOUR ANSWER BELOW

def reverse\_zip(zip\_obj):

"""

Given a zip\_object, return the lists passed into `zip()` to create

That zip\_object

Positional Argument:

zip\_obj -- a zip object

Example:

zip\_obj = zip([5,6],[9,10],["a","b"])

print(reverse\_zip(zip\_obj)) #--> [[5,6],[9,10],["a","b"]]

"""

return []

1. Response
2. **Question**

### GRADED

### Code a function called 'obj\_indicies'

### ACCEPT Two Inputs: a list, then some object to search for.

### RETURN a list of the indicies where that object appears in the list.

### e.g. ([1,2,2,3], 2) should return [1,2]; ([1,2,2,3], 4) should return []

### YOUR ANSWER BELOW

def obj\_indicies(list\_to\_search, search\_for):

"""

Return a list of indicies which are all the locations of the specified

object in the passed list

Positional Arguments:

list\_to\_search - a list of objects

search\_for - the object to be searched for.

Example:

l1 = [1,2,2,3,4,5,10]

print(obj\_indicies(l1,2)) #--> [1,2]

print(obj\_indicies(l1,6)) #-->[]

print(obj\_indicies(l1,10)) #-->[6]

"""

Pass

1. Response
2. **Question**

### GRADED

### Code a function called "return\_value"

### ACCEPT two inputs: a dictionary and a key from that dictionary

### ### ASSUME the provided key is in the dictionary

### RETURN the value associated with that key in the dictionary

### YOUR ANSWER BELOW

def return\_value(input\_dict, input\_key):

"""

Return the value from the inputted dictionary located at the location of

the given key

Positional Arguments:

input\_dict - a dictionary

input\_key - a key in that dictionary

Example:

test\_dict = {1:2, "A":"B", 3:"c", "1":"a"}

print(return\_value(test\_dict,"A")) #--> "B"

print(return\_value(test\_dict,3)) #--> "c"

print(return\_value(test\_dict,"1")) #--> "a"

"""

Pass

1. Response
2. **Question**

### GRADED

### This exercise involves building a non-trivial dictionary.

### The subject is books.

### The key for each book is its title

### The value associated with that key is a dictionary

### ### In that dictionary there will be Three keys: They are all strings, they are:

### ### "Pages", "Author", "Publisher"

### ### ### "Pages" is associated with one value - an int

### ### ### "Author is associated with a dictionary as value

### ### ### ### That "Author" dictionary has two keys: "First", and "Last" each with a string value

### ### ### "Publisher" is associated with a dictionary as value

### ### ### ### That "Publisher" dict has one key "Location" with a string as value.

### An Example might look like:

### {"Harry Potter": {"Pages":200, "Author":{"First":"J.K", "Last":"Rowling"}, "Publisher":{"Location":"NYC"}},

### "Fear and Lothing in Las Vegas": { ...}}

### Code a function called "build\_book\_dict"

### ACCEPT five inputs, all lists of n-length

### ### A list of titles, pages, first<name>, last<name>, and <publisher>location.

### RETURN a dictionary as described above.

### Keys must be spelled just as they appear above - correctly and capitalized.

### YOUR ANSWER BELOW

def build\_book\_dict(titles, pages, firsts, lasts, locations):

"""

Return a nested dictionary storing information about Books

Positional Arguments:

titles -- A list of strings

pages -- A list of ints

firsts -- A list of strings

lasts -- A list of strings

locations -- A list of strings

Example:

titles = ["Harry Potter", "Fear and Lothing in Las Vegas"]

pages = [200, 350]

firsts = ["J.K.", "Hunter"]

lasts = ["Rowling", "Thompson"]

locations = ["NYC", "Aspen"]

book\_dict = build\_book\_dict(titles, pages, firsts, lasts, locations)

print(book\_dict) # -->

{'Fear and Lothing in Las Vegas': {'Publisher': {'Location': 'Aspen'},

'Author': {'Last': 'Thompson', 'First': 'Hunter'}, 'Pages': 350},

'Harry Potter': {'Publisher': {'Location': 'NYC'},

'Author': {'Last': 'Rowling', 'First': 'J.K.'}, 'Pages': 200}}

"""

return dict()

1. Response
2. **Question**

### GRADED

### In which of the following is the "\*" NOT used in Python?

### 'a') multiplication

### 'b') exponents

### 'c') tuple unpacking

### 'd') passing dictionaries into functions as parameters

### 'e') if statement syntax

### 'f') passing lists into functions for parameter

### Assign string associated with your choice to ans1

### YOUR ANSWER BELOW

ans1 =''

1. Response
2. **Question**

### GRADED

### Code a function called "divisible\_by\_3"

### ACCEPT a list of numbers (int or float) as input

### RETURN a list of all the numbers in that list that are divisible by 3, multiplied by 2.

### e.g. [1,2,3,4,5,6] as input should yield [6,12]

### ### Preferably use a list comprehension

### YOUR ANSWER BELOW

def divisible\_by\_3(input\_list):

"""

Return all of the numbers that were divisible by 3 multiplied by 2

Positional Arguments:

input\_list -- A list of numbers, floats or ints

Example:

num\_list = [1,4,5,6,7,8,2,9,3,3,6,9]

print(divisible\_by\_3(num\_list)) #--> [12, 18, 6, 6, 12, 18]

"""

return []

1. Response
2. **Question**

### GRADED

### Code a function called "final\_element"

### ACCEPT a nested list as an input.

### Input will be in the format [[1,2,3,...,n],[4,5,6,...,n],...,['x','y','z',...,n-1,'q']]

### More specifically, "m" lists, all of "n" elements, will be contained in the nested list

### RETURN the final element in the final list from that nested list.

### In the above example, that would be the string 'q'.

### YOUR ANSWER BELOW

def final\_element(input\_nested\_list):

"""

Return the final element in the final list of the input

Positional Argument:

input\_nested\_list - a list containing "m" lists,

each of which has "n" elements

Example:

nested = [[1,2,3],[4,5,6],[7,8,"a"]]

print(final\_element(nested)) #--> 'a'

"""

return ''

1. Response
2. **Question**

### GRADED

### Code a function called "apply\_functions"

### ACCEPT two lists as inputs:

### The first list will be a list of iterables

### The second list will be some subset of the functions <len>, <sum>, and <type>

### RETURN a list of the same length as the first list, where every element in that list

### is a \*tuple\* which has had the functions from the second list applied on them.

### e.g. if the lists are [(1,2),[3,4]], and [len, sum, type] The return should be:

### [(2,3,tuple),(2,7,list)].

### Note the tuples are in the order of (length, sum, type); the same order as the passed functions

### YOUR ANSWER BELOW

def apply\_functions(list\_of\_objs, list\_of\_funcs):

"""

Return a list where all the functions in the second argument

have been applied to the elements of the first argument

Positional Arguments:

list\_of\_objs -- A list of objects (lists and/ or tuples)

list\_of\_funcs -- A list of functions that may be applied to the

objects of the first argument

Example:

objs = [(1,2),[1,3,4,5,6,7],[0]]

funcs = [len,sum]

print(apply\_functions(objs, funcs))

#--> [(2, 3), (6, 26), (1, 0)]

"""

return []

1. Response
2. **Question**

### GRADED

### In which of the following is the "\*" NOT used in Python?

### 'a') multiplication

### 'b') exponents

### 'c') tuple unpacking

### 'd') passing dictionaries into functions as parameters

### 'e') if statement syntax

### 'f') passing lists into functions for parameter

### Assign string associated with your choice to ans1

### YOUR ANSWER BELOW

ans1 =''

1. Response